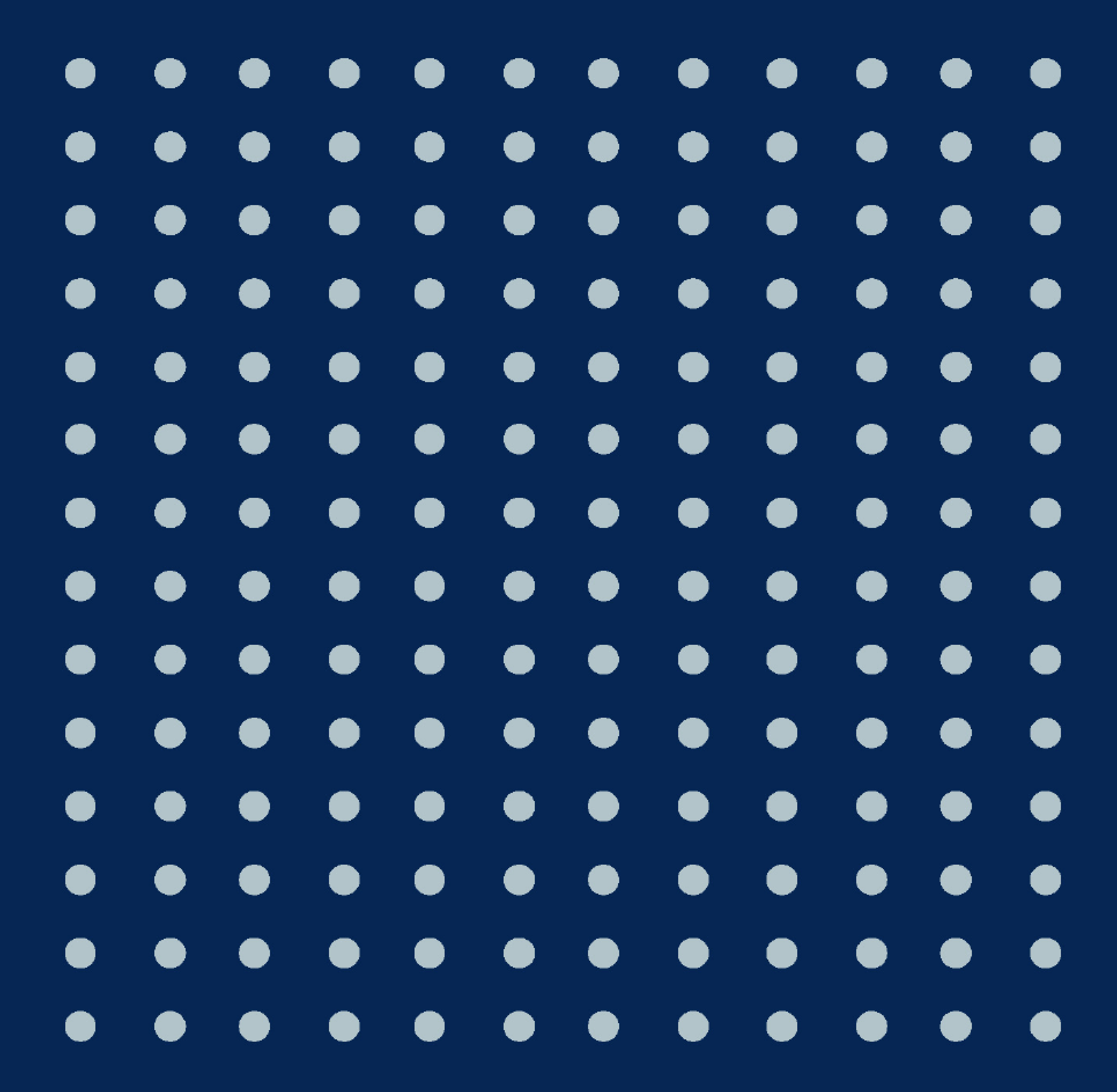
重点大学计算机专业系列教材

对等网络、网络计算与云计算

——原理与安全

邹福泰 张亮 陈曙东 王旹 编著



清华大学出版社

重点大学计算机专业系列教材

对等网络、网格计算与云计算
——原理与安全

邹福泰　张亮　陈曙东　王岢　编著

清华大学出版社
北京

内 容 简 介

本书详细阐述对等网络、网格计算和云计算的基本原理和安全技术，内容丰富，系统性强。全书共分10章，分别是计算模式的演变、对等网络研究进展、对等网络拓扑及优化、对等网络信息检索、网格计算研究进展、网格资源管理体系结构、网格资源调度算法研究、云计算文件系统设计与实现、计算安全、总结与展望。

本书适合作为高等院校计算机应用、计算机网络、通信与信息系统、电子与信息工程等相关专业的本科生和研究生教材，也适合以上相关专业的研究开发人员和工程技术人员参考。

图书在版编目（CIP）数据

对等网络、网格计算与云计算：原理与安全/邹福泰等编著.--北京：清华大学出版社，2012.8
（重点大学计算机专业系列教材）
ISBN 978-7-302-28821-3

Ⅰ.①对…　Ⅱ.①邹…　Ⅲ.①计算机网络－高等学校－教材　Ⅳ.①TP393

中国版本图书馆CIP数据核字(2012)第102673号

责任编辑：高买花　张为民
封面设计：
责任校对：李建庄
责任印制：

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦A座　**邮　　编**：100084
社 总 机：010-62770175　**邮　　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课件下载：http://www.tup.com.cn，010-62795954
印 刷 者：
装 订 者：
经　　销：全国新华书店
开　　本：185mm×260mm　**印　　张**：16.25　**字　　数**：406千字
版　　次：2012年8月第1版　**印　　次**：2012年8月第1次印刷
印　　数：1～　000
定　　价：　.00元

产品编号：039761-01

INTRODUCTION

出版说明

随着国家信息化步伐的加快和高等教育规模的扩大，社会对计算机专业人才的需求不仅体现在数量的增加上，而且体现在质量要求的提高上，培养具有研究和实践能力的高层次的计算机专业人才已成为许多重点大学计算机专业教育的主要目标。目前，我国共有16个国家重点学科、20个博士点一级学科、28个博士点二级学科集中在教育部部属重点大学，这些高校在计算机教学和科研方面具有一定优势，并且大多以国际著名大学计算机教育为参照系，具有系统完善的教学课程体系、教学实验体系、教学质量保证体系和人才培养评估体系等综合体系，形成了培养一流人才的教学和科研环境。

重点大学计算机学科的教学与科研氛围是培养一流计算机人才的基础，其中专业教材的使用和建设则是这种氛围的重要组成部分，一批具有学科方向特色优势的计算机专业教材作为各重点大学的重点建设项目成果得到肯定。为了展示和发扬各重点大学在计算机专业教育上的优势，特别是专业教材建设上的优势，同时配合各重点大学的计算机学科建设和专业课程教学需要，在教育部相关教学指导委员会专家的建议和各重点大学的大力支持下，清华大学出版社规划并出版本系列教材。本系列教材的建设旨在“汇聚学科精英、引领学科建设、培育专业英才”，同时以教材示范各重点大学的优秀教学理念、教学方法、教学手段和教学内容等。

本系列教材在规划过程中体现了如下一些基本组织原则和特点。

1. 面向学科发展的前沿，适应当前社会对计算机专业高级人才的培养需求。教材内容以基本理论为基础，反映基本理论和原理的综合应用，重视实践和应用环节。

2. 反映教学需要，促进教学发展。教材要能适应多样化的教学需要，正确把握教学内容和课程体系的改革方向。在选择教材内容和编写体系时注意体现素质教育、创新能力与实践能力的培养，为学生知识、能力、素质协调发展创造条件。

3. 实施精品战略，突出重点，保证质量。规划教材建设的重点依然是专业基础课和专业主干课；特别注意选择并安排了一部分原来基础比较好的优秀教材或讲义修订再版，逐步形成精品教材；提倡并鼓励编写体现重点大学

计算机专业教学内容和课程体系改革成果的教材。

4. 主张一纲多本，合理配套。专业基础课和专业主干课教材要配套，同一门课程可以有多本具有不同内容特点的教材。处理好教材统一性与多样化的关系；基本教材与辅助教材以及教学参考书的关系；文字教材与软件教材的关系，实现教材系列资源配套。

5. 依靠专家，择优落实。在制订教材规划时要依靠各课程专家在调查研究本课程教材建设现状的基础上提出规划选题。在落实主编人选时，要引入竞争机制，通过申报、评审确定主编。书稿完成后要认真实行审稿程序，确保出书质量。

繁荣教材出版事业，提高教材质量的关键是教师。建立一支高水平的以老带新的教材编写队伍才能保证教材的编写质量，希望有志于教材建设的教师能够加入到我们的编写队伍中来。

教材编委会

FOREWORD

前言

从 2000 年开始，对等网络（Peer-to-Peer Network，P2P Network）一直是计算机和互联网领域最受关注的热门话题之一。《财富》杂志将 P2P 技术列为影响 Internet 未来的四项科技之一，著名的英特尔公司也给了 P2P 技术极高的评价，称它为“第三代网络革命”。对等网络将分布于世界各地的个人计算机组织起来，通过交换进行资源和服务的共享。这些资源和服务包括信息交换、高速缓存、处理能力、存储空间等。在对等网络中，每个结点自治又彼此依赖，自治是指每个结点独立决定自己的行为而不受其他如集中式授权机构的控制，同时每个结点又需要相互协作获得信息资源、计算资源等。对等网络具有自组织特性，表现为网络具有高度的拓扑弹性和容错性。通常一个对等网络规模可达到几十万甚至上百万个计算结点，参与的计算结点以一种松散方式进行组织，非常适合广域网络和 Internet 的应用，并且已经涌现了不少非常具有影响力的应用系统，如 PPLive、PPStream、BT、eMule、Skype 等。事实上，对等网络已经在文件共享与内容分发、分布式数据存储、分布式计算、协同工作与服务共享、分布式深度搜索引擎、即时通信以及应用层组播等多个领域得到了广泛应用。对等网络不仅是一种技术体系，更是一种思想变革。它深刻影响了 Internet 的诸多应用，被视为下一代 Internet 应用的基础。

随着计算机技术和 Internet 网络技术的发展，一方面，随着网络带宽的不断拓宽和计算机处理能力的不断增强，很多计算机资源的利用效率远没有得到发挥；另一方面，由于互联网上的信息资源的不断增长，即便是 Google 这样功能强大的搜索引擎也只能搜索极少部分的内容。网格技术就如何更好地利用现有各种计算机软硬件资源和网络信息资源提出了一个很好的解决思路。20 世纪 90 年代中期，网格首次被提出用于描述科学和工程分布式计算的基础设施，它把计算资源、数据存储设施、广域网络、仪器设备等连成有机的整体，方便用户使用这个基础设施中的任何资源。网格系统的构建需要研究信息服务、数据管理、安全机制等技术。另外，网格资源的动态性、异构性和自治性，使得网格资源提供者和使用者具有对等网络的特点，因此有必要将对等网络方法引入到网格资源管理中。为此，本书基于对等网络技术提出了一系列网格资源管理和调度策略。

云计算概念是由Google提出的，这是一个美丽的网络应用模式。广义上的云计算是指服务的交付和使用模式，指通过网络以按需、易扩展的方式获得所需的服务。这种服务可以是IT和软件、互联网相关的，也可以是任意其他的服务，它具有超大规模、虚拟化、可靠安全等独特功效。云计算的出现为信息技术领域带来了新的挑战，也为信息技术产业带来了新的机遇。云计算的目标是将计算和存储简化为像公共的水和电一样易用的资源，用户只要连上网络即可方便地使用，按量付费。云计算提供了灵活的计算能力和高效的海量数据分析方法，企业不需要构建自己专用的数据中心就可以在云平台上运行各种各样的业务系统，这种创新的计算模式和商业模式吸引了产业界和学术界的广泛关注。本书在介绍云计算相关概念的基础上，详细研究探讨了云计算相关的分布式文件系统(DFS)架构，并针对云计算平台的安全问题进行深入探讨，结合虚拟化技术对云计算平台的安全性进行了研究。

有感于对等网络、网格计算和云计算技术的迅速发展，以及研究开发人员对此领域进一步研究开发的需求，作者在自身研究工作积累的基础上精心编写了本书，让读者分享我们学习与研究工作的经验和成果。本书不仅可以使初学者能够了解领域内的研究现状，也可以为有一定研究基础的同行提供较为系统的相关的技术或方案，弥补这一领域内国内研究资料相对匮乏的境况，缩短国内学者的研究水平与国际一流水平的差距，贡献出更多的具有自主知识产权的研究成果。

本书共分10章，各章主要内容介绍如下：

第1章介绍计算模式演变。本章首先介绍计算模式的发展历程，详细分析了过去几年人们先后提出的C/S计算、B/S计算、网格计算、服务计算、对等计算、云计算等计算模式，分析了网格计算和云计算的区别，最后阐述了本书的研究脉络以及层次结构。

第2章介绍对等网络研究进展。对等网络系统是一个新兴的研究领域，近些年得到了迅速发展。本章首先对对等网络进行了概述性介绍，然后重点介绍了对等网络在路由、拓扑和查询这三方面的研究工作和研究进展。

第3章讨论对等网络拓扑及优化。本章首先针对DHT拓扑模型在动态网络中性能下降问题，提出了一种改善的SHT模型。SHT模型利用对等网络结点存在的会话异构性，将动态结点聚簇在稳定结点，从而降低网络动态结点对于DHT拓扑的强干扰性。仿真实验显示了SHT模型能够有效减少对等网络系统的拓扑维护开销以及提高系统稳定性和数据可用性。然后，基于小世界模型理论提出了可应用于对等网络路由优化的一种概率缓存链模式，通过理论分析以及在静态和动态网络的实验，证明了此模式在附加少量开销的情况下，对DHT路由的性能具有较大的改善。最后，对Chord算法的最优路由进行深入分析，提出Chord协议的最优路由表结构，给出基于最优路由表结构的路由算法，并证明Chord在满环情况下三倍数路由表构造是最优路由表结构。在Chord不满环的情况下，对路由表进行重构，引入多种重构策略，并对其进行仿真实验，证明在重构后的路由算法切实有效。

第4章讨论对等网络信息检索。本章首先介绍常用的信息检索所涉及的分类算法，对P2P文件的索引(发布)和用户模式确定所涉及的分类问题进行了研究，提出了基于分类器融合的文本分类方法m-SLMBSVM-kNN。该方法将SLMBSVM方法与kNN方法结合起来，通过两个分类器在不同空间的分类行为，使各个分类器可以形成优势互补，从而提高超文本分类的精度。接下来介绍多关键字检索所涉及的相关技术和研究现状，然后针对于目前多关键字检索系统的不足，提出基于检索关键词关联的关键词集发布的多关键词检索系统。

第5章介绍网格计算研究进展。本章分析网格计算的研究重点和研究目的，即为了更好地进行网格资源管理和调度，实现网格系统的实时动态调度和负载均衡；说明现有的网格资源管理和调度策略的基本原理以及在动态环境中的局限性，进而指出进行网格动态资源管理和调度策略研究的背景和动机。本章还介绍网格资源管理体系结构的基本概念和主要研究内容，分析目前的研究存在的一些问题，最后阐述本书网格计算部分的主要研究内容和创新点。

第6章讨论网格资源管理体系结构。本章首先介绍网格发展的新特征，这些特征是本书提出的模型和算法的一个重要基础；对现有网格资源管理采用的基本体系结构、模型做了详细的分析，指出它们存在的问题；在分析现有资源管理体系结构由于不能反映网格资源的动态性和自治性而影响资源调度效果的基础上，提出基于对等网络技术的混合资源管理模型，该模型可以使得网格系统具有很好的鲁棒性和容错性。最后介绍使用有向图进行基于超级结点对等网络的网格系统 overlay network 拓扑表示的方法。

第7章分析网格资源调度算法。本章分四部分对网格资源调度算法进行研究。第一部分设计一个专用计算网格环境中的实时资源调度算法 nTreeMatch。该算法利用树匹配完成资源和任务间的映射，并且通过充分利用 overlay network 拓扑提供的路由信息来提高算法调度效率，降低 RDP。最后通过理论分析和大量的实验证明基于树匹配的 nTreeMatch 算法对于提高网格动态资源调度是非常有效的。第二部分分析现有资源管理方法均采用信息收集的方式，结合网格的高度动态性特征，指出可以将资源管理和调度问题转化为服务发现问题，然后在分析现有的集中式服务发布和发现方法的基础上，提出适用于基于 Web Service 的通用网格系统环境的基于资源发现的资源管理和调度算法 GChord，最后通过理论分析和对比实验证明该方法的有效性。第三部分研究网格结点的自治性特点以及多代理技术的基本原理和应用，指出两个研究领域的相似性和互通性，可以借鉴代理的智能学习弥补现有网格系统很难实现动态调度和负载均衡的缺点，提出一个基于多代理协同计算的网格资源调度算法 rwAgent。最后通过严格的数学建模和实验证明 rwAgent 算法可以实现网格系统的实时资源调度，并且可以获得好的负载均衡效果。第四部分介绍 DDGrid——新药研发网格的框架结构以及系统平台的实现。它的实现充分利用了本章的研究成果。DDGrid 接受用户从 Web 上提交的药物筛选计算任务，自动取出和计算相关的数据项，进行并行任务划分，然后将这些计算任务映射到网格系统的各计算结点中。DDGrid 利用中国国家网格各计算中心的集群提供的空闲计算力，进行虚拟药物筛选实验计算，在计算结束后，DDGrid 会通知用户通过 Grid Portal 下载计算结果。用户只需要提交药物筛选所需的原始文件，DDGrid 便可自动地进行处理，此过程对于用户完全透明。

第8章介绍云计算文件系统设计与实现。本章首先介绍云计算的基本概念及关键技术，然后对云存储系统架构进行分析，最后介绍云计算文件系统（CFS）的设计与实现。分布式文件系统(DFS)是一种网络文件系统，它分布在多个计算机结点上，每个结点只会直接存取整个文件系统的一部分。本章参考两个比较成熟的 DFS 架构——GFS 和 Hadoop，设计并实现 CFS。CFS 是一个面向数据密集型应用的分布式文件系统，整个系统由 Master 结点、数据结点和客户端组成，具有一定的容错性和可扩展性。

第9章讨论计算安全。本章首先针对对等网络的结点的自私行为和恶意行为，介绍当前对等网络的激励机制、信任机制以及文件安全机制的研究工作。接下来，针对云计算平台

的安全问题进行深入探讨，结合虚拟化技术对云计算平台的安全性进行研究。

第10章是总结与展望。本章介绍三种计算模式将来进一步研究的内容和方向。

本书在编写过程中得到上海交通大学信息安全工程学院李建华教授的关心与支持，以及上海交通大学计算机科学与工程系马范援教授的悉心指导，在此向他们表示衷心的感谢。同时衷心感谢清华大学出版社的大力支持，感谢编辑为本书付出的辛勤劳动。

对等网络、网格计算与云计算涉及领域宽、发展快，本书取材于学术界和工程技术界的一些研究成果，也包括作者的一些成果和观点。相关研究成果属于其他原创者，我们在书中均作了引用标识。我们尽量以客观的态度对待任何一项研究方法和成果，对于其中的争议甚至错误，希望留待读者去进一步甄别与探究。尽管我们力求完美，但作者水平有限，本书疏漏、不当与错误之处在所难免，欢迎读者批评指正。

本书得到国家自然科学基金(编号：61078011)、国家高技术研究发展863计划(编号：2002AA104270,2009AA01Z118)、上海市科委科技攻关重大项目(编号：03DZ15027)等项目的资助。

作　者

2012年6月于上海交通大学

CONTENTS

目录

CHAPTER 1

第1章 计算模式的演变

信息技术的高速发展推动了计算模式的不断更新。从单机时代的主机/终端模式、文件服务器时代的共享数据模式、客户机/服务器时代的C/S计算模式到万维网时代的B/S网络计算模式，再到目前的P2P对等计算模式以及网格计算模式和云计算模式，计算模式已经发生了巨大变化。

1.1 计算模式

20世纪90年代，计算技术最引人注目的进展之一就是应用计算环境从集中走向分布，其中，客户机/服务器(Client/Server，C/S)计算技术成为分布式计算的主流技术。随着技术的发展，基于C/S计算模式构建的商用计算体系结构表现出愈来愈大的局限性，它的网络优势局限于企业内部，难以突破企业之间的组织边界，企业与企业之间的信息交流受到很大制约，同时它还存在企业软件购置开销过大、网络安全性较差等诸多不足之处。这使得C/S计算模式已不能适应更高速度、更大地域范围的数据运算和处理。全球化、协作化、个性化决定了浏览器/服务器(Browse/Server，B/S)计算模式的出现。但这种模式要求在互联网上设置拥有强大处理能力和高带宽的高性能计算机。计算机需安装有高档的服务器软件，再将大量的数据集中存放在上面，并且还要安装多样化的服务软件，在集中处理数据的同时其可以对互联网上其他PC进行服务。这种计算模式的缺点是没有有效地利用客户端的资源，包括处理器、存储器、文件内容等。后来又出现了一种计算模式，即对等Peer-to-Peer，P2P计算模式。P2P计算技术的出现目的就是希望能够充分利用互联网中所蕴含的潜在计算资源。P2P计算技术的特征之一就是弱化了服务器的作用，甚至取消服务器，任意主机既是服务器，同时又是客户机，即对等。对等网络实现了从互联网的最终使用者到主动的服务提供者的转变。

在C/S计算模式下，数据通常存储在有固定结构的数据库中；在B/S计算模式下，数据以半结构化的网页形式表现出来；在P2P计算模式下，数据的表现形式变成了无结构、多样化的文件。在C/S计算模式下，随着数据量

的进一步增大,人们面临着数据爆炸但知识匮乏的尴尬境界,为了有效地解决这一问题,面向数据库的数据挖掘应运而生。在 B/S 计算模式下,随着 Web 页面数量的迅速增加,Internet 成为目前最大的知识库,对其进行模式和知识的获取已经成为必然趋势,因而面向 Web 挖掘(Web Mining)应运而生。在 P2P 计算模式下,随着加入 P2P 网络的对等结点的迅速增加以及发布文件的急剧增多,在 P2P 网络中如何获取自己所需要的东西已经变得越来越困难,这时候,针对于 P2P 文件的模式和知识的对等网络挖掘(P2P Mining)也就应运而生。

随后出现的网格(Grid)计算模式是借鉴电力网(Electric Power Grid)概念提出的,网格计算的最终目标是实现用户在使用计算力时,能够如同当前使用电能一样方便。当人们在使用电能时,无须知道它的来源地以及产生的方式,只要知道它是一种统一形式的电能即可。类似地通过网格计算,用户希望在使用计算力时,无须关心计算力的来源地以及计算设施的形式,计算力以统一的形式呈现在用户面前。网格计算代表了一种先进的技术和基础设施,是分布式高性能计算和高吞吐量计算的主要发展方向。

目前正热的云计算模式是此前并行计算(Parallel Computing)、分布式计算(Distributed Computing)、对等计算和网格计算(Grid Computing)的发展,或者说是这些计算机科学概念的商业实现。云计算是虚拟化(Virtualization)、效用计算(Utility Computing)、基础设施即服务(IaaS)、平台即服务(PaaS)、软件即服务(SaaS)等概念混合演进并跃升的结果。

1.2 主机/终端计算

20 世纪 60—70 年代的主机计算模式是第一代计算模式,由大型机和多个与之相连的哑终端组成。当时的计算模式为主机/终端模式(即集中式计算模式),运行在主机上的 UNIX 操作系统是一个多用户、多任务和多进程的操作系统,用户终端仅仅是一个输入输出接口。由于物理设备的限制,采用这种计算模式的所有计算数据和程序都只能在主机系统上,从而形成典型的"集中存储、集中计算"模式。

1.3 C/S 和 B/S 计算

随着计算机网络化技术的兴起,计算模式也在发生变化。

20 世纪 80—90 年代中期是第二代计算模式,其显著特征是计算机应用领域被大大拓宽,桌面办公应用和数据库技术的大力发展,导致协同计算和分布式计算理念迅速蔓延。特别随着 C/S 计算模式的兴起,似乎"分布"成了解决异种结构和平台之间资源共享的最佳方法。在 C/S 计算模式中,服务器只负责各种数据的处理和维护,为各个客户机应用程序管理数据;客户机包含文档处理软件、决策支持工具、数据查询等应用逻辑程序,通过网络使用 SQL 语言发送、请求和分析从服务器接收的数据。这是一种"胖客户机"(Fat Client)、"瘦服务器"(Thin Server)的网络结构模式。然而,分布的理念在管理和兼容方面存在不足:整个网络系统和客户机系统维护、软件安装烦琐,大规模系统很难进行有效的维护,而且在分布式系统上保持数据同步是一项极为复杂的任务。

为解决传统的两层式 C/S 体系结构中的这些固有问题，便出现了三层或多层的计算模式，即客户机/中间服务器/数据库服务器（应用服务器）模式。在这种三层式的体系结构中，客户机只完成基本的显示、输入和输出，或者一些简单的事物处理；中间服务器插入到传统的 C/S 体系中，完成由原客户机所进行的数据前后处理工作，并负责为其前台的客户机提供显示服务，同时与后台的一个或多个数据库服务器进行大量的数据传递。客户机相对于中间服务器来说是一个 C/S 关系。随着网络和计算机技术的发展和多用户 Windows NT 操作系统的问世，这种三层体系的代表——Thin-Client/Server 计算模式，在众多的领域中得到了越来越广泛的应用。

多层结构计算模式是把业务逻辑独立出来，组成一层或多层，形成客户层、中间业务处理层（可由多层组成）和后端数据服务层。在多层结构中，层次的划分不是物理上的划分，而是结构逻辑上的划分，按应用目标划分。如果客户端要求响应速度很快，业务组件的体积较小，业务组件可以放在客户端；如果业务组件包含大量对数据库的操作，可以配置在数据库服务器上，以减少网络负载，提高运算速度；如果业务组件可供大多数客户机程序访问，则可以使用业务组件构成一个应用服务器供访问。

B/S 计算模式是一种从传统的二层 C/S 计算模式发展起来的新的网络结构模式，其本质是三层结构 C/S 计算模式。在 B/S 计算模式中，客户端运行浏览器软件。浏览器以超文本形式向 Web 服务器提出访问数据库的要求，Web 服务器接受客户端请求后，将这个请求转化为 SQL 语法，并交给数据库服务器，数据库服务器得到请求后，验证其合法性，并进行数据处理，然后将处理后的结果返回给 Web 服务器，Web 服务器再一次将得到的所有结果进行转化，变成 HTML 文档形式，转发给客户端浏览器以友好的 Web 页面形式显示出来。B/S 网络结构模式是基于 Intranet 的需求而出现并发展的。

最初的集中式大型机结构所采用的传统终端相对其主机而言是一种“瘦型”客户端，然后经历过功能强大的“胖型”客户端即 PC 之后，却又一次回到“瘦型”客户端。但这并非是一种简单的循环往复，而是计算机体系结构的发展向“开放式”结构不断迈进的过程。计算模式始于非常封闭的集中式主机，正在向一个真正开放的、与平台完全无关的体系过渡。

1.4　分布式计算

分布式计算是一门计算机科学，它研究如何把一个需要非常巨大的计算能力才能解决的问题分成许多小的部分，然后把这些部分分配给许多计算机进行处理，最后把这些计算结果综合起来得到最终的结果。最近的分布式计算项目已经被用于使用世界各地成千上万位志愿者的计算机的闲置计算能力，通过 Internet，可以分析来自外太空的电信号，寻找隐蔽的黑洞，并探索可能存在的外星智慧生命；可以寻找超过 1000 万位数字的梅森质数；也可以寻找并发现对抗艾滋病病毒的更为有效的药物。这些项目都很庞大，需要惊人的计算量，仅由单一的计算机或是个人在一个能让人接受的时间内计算完成是绝对不可能的。

分布式计算就是在两个或多个软件之间互相共享信息，这些软件既可以在同一台计算机上运行，也可以在通过网络连接起来的多台计算机上运行。分布式计算比起其他算法具有以下几个优点：

（1）稀有资源可以共享。

(2) 通过分布式计算可以在多台计算机上平衡计算负载。

(3) 可以把程序放在最适合运行它的计算机上。

其中,共享稀有资源和平衡计算负载是计算机分布式计算的核心思想。

1.5 网格计算

网格计算的概念最初由 I-WAY[1] 项目在 1995 年提出。其前身是元计算[2](Metacomputing)。早期的元计算被定义为在一个网络环境下使用户透明地获得强大的计算资源,过去对于元计算的研究可以认为是网格计算的初级阶段。

关于网格和网格计算本身还没有一个确切的定义。根据 Ian Foster 博士早期的定义[3],网格是一个集成的计算与资源环境,或者说是一个计算资源池,网格能够充分吸纳各种计算资源,并将它们转化为一种随处可得的、可靠的、标准的、经济的计算能力。除了各种类型的计算机外,这里的计算资源还包括网络通信能力、数据资料、仪器设备等资源。但随着近 10 年的发展,网格计算的研究重点发生了转移,在参考文献[4]中,网格被定义为:基于 Internet 技术和 Web 技术,采用开放式标准,为动态参与的多个机构组成的虚拟组织完成某类科学、工业或工程上的应用提供可扩展、安全、高效、灵活、协调的共享资源。网格计算关心的是在动态的、多机构的虚拟组织中协调资源共享和协同解决问题。针对网格概念模糊现象,在参考文献[5,6]中,Ian Foster 阐述了网格与目前网络的区别,也是判断一个系统是否可以被称为网格的标准。

网格的特点如下:

(1) 建立在现有的 Internet 技术和分布计算技术之上。

(2) 采用标准、开放、统一的接口和协议。通过这类接口和协议以解决广域范围内的认证、授权、资源查找和定位、资源访问等问题。

(3) 资源共享。对分布在网络上的高性能计算机、软件、数据、大型数据库、可视化设备、贵重仪器(电子显微镜、粒子加速器、天文望远镜等)及其他设备均可共享。与计算机网络不同,计算机网络实现的是硬件的连通,与目前的网络服务也不同,目前的网络服务仅实现了网页的连通,而网格能实现应用层面全方位的连通。

(4) 非集中控制。网格中的资源分散在多个控制域内,归属于不同的机构,有不同的安全机制、计费方式和管理方法,网格中的资源是通过协商而不是强占进行共享。

(5) 提供有保证的服务质量。网格可以根据用户的权限和要求提供不同质量的服务,服务质量可以用响应时间、吞吐率、可用性、安全性等要素来衡量。

网格发展至今,以 Globus Toolkits[7] 为基础的计算网格日趋成熟,基本可以保证一定的服务质量和安全性,通过对多级管理域下集中控制的超级计算资源的共享,可以在计算网格上实施多样化的大规模并行计算。

网格还具有以下特点:

(1) 广域分布。参与计算的资源不仅存在于局域网内,它们分布在世界的各个区域。广域分布的特点使得应用程序的设计要考虑容忍相当大的网络延时、通信链路的竞争、路由等。

(2) 异构性。参与计算的计算机在软件工具、硬件设备上不同,即使软件、硬件相同,在

不同时间段表现的性能特性也不同，而且这些计算资源存在于不同的管理域中。

(3) 动态性。由于多用户特别是资源不可获取等情况的存在，资源的可用性、用户的目标和优先次序等都呈现动态变化。

网格的这些独有特性使得网格系统比以往的分布式系统更为复杂。

最新的网格研究[4]希望通过对大规模分散资源的共享，为动态组建的虚拟组织提供有保证的服务质量。因此，网格未来的发展趋势将具备以下特点：

(1) 与对等计算相融合。P2P 计算被称为对等计算。P2P 系统中的各个结点不依赖于特定的集中式机制，而是因为互为服务而共存，各结点可以直接交互并可能随时离开 P2P 网络。网格中的服务提供者可以动态加入或者退出。

(2) 协同共享大规模异构资源。异构性可以体现在如下两个方面：

① 多种资源。资源可以是不同类型的，每种资源的服务能力是不同的，如计算资源、数据资源、服务、存储空间等。

② 多种特性。某些资源提供的服务是稳定的，如"从早上 6 点到下午 6 点为所有用户服务"；而有些资源提供的服务是不稳定的，如"只有在空闲时资源可被共享"。

(3) 多种应用模式。结合网格服务技术，未来的资源共享方式是标准和通用的。

(4) 集中管理和非集中管理控制并存。从对外服务的角度看，网格是集中管理的，从虚拟组织内部的角度看，资源加入退出的模式是高度可变的，无须集中控制和集中验证。

1.6　服务计算

服务计算是跨越计算机与信息技术、商业管理、商业质询服务等领域的一个新的学科，是应用面向服务体系架构(Service Oriented Architecture，SOA)技术在消除商业服务与信息支撑技术之间的横沟方面的直接产物。它在形成自己独特的科学与技术体系的基础上有机整合了一系列最新技术成果：SOA、Web 服务、网格/效用计算(Grid & Utility Computing)以及业务流程整合及管理(Business Process Integration & Management)。第一部分解决的是技术平台和架构的问题，第二部分解决的是服务交付的问题，第三部分则是业务本身的整合和管理。

1.7　云计算

随着多核处理器、虚拟化、分布式存储、宽带互联网和自动化管理等技术的发展，产生了一种新型的计算模式——云计算，即用户终端可以通过远程网络连接，获取存储、计算、数据库等计算资源。现今，云计算是一个很热门的话题，可以说我们即将要进入了云计算的时代。人们重视云计算的原因，一个是因为云计算这种商业模式孕育着巨大的商业机遇，另一方面，云计算作为互联网资源的新的配置方式，它将改变整个互联网的价值链。

众所周知，云计算是一种服务，其提供无所不在、无时不在的互联网信息服务。它彻底改变了人们使用信息服务的方式，使人们摆脱了为了使用信息必须学会使用复杂的信息技术和设施的负担，使人们更加关注信息和内容的本身，而不是其他。

"云"由分布的互联网基础设施(网络设备、服务器、存储设备、安全设备等)等构成，几乎

所有的数据和应用软件，都可存储在“云”里。“云终端”如 PC、手机、车载电子设备等，只需要拥有一个功能完备的浏览器，并安装一个简单的操作系统，通过网络接入“云”，就可以轻松地使用云中的计算资源。

云计算的核心是按需部署，因而需要解决资源的动态可重构、监控和自动化部署等问题，而解决这些问题又需要以虚拟化技术、高性能存储技术、处理器技术、高速互联网技术为基础。所以云计算除了需要仔细研究其体系结构外，还要研究资源的动态可重构、自动化部署、资源监控、虚拟化技术、高性能存储技术、处理器技术等。

云计算从概念上分为狭义云计算和广义云计算。狭义云计算是指 IT 基础设施的交付和使用模式，指通过网络以按需、易扩展的方式获得所需的资源（硬件、平台、软件）。提供资源的网络被称为“云”。“云”中的资源在使用者看来是可以无限扩展的，并且可以随时获取，按需使用，随时扩展，按使用情况付费。就像使用水电一样使用 IT 基础设施。广义云计算是指服务的交付和使用模式，指通过网络以按需、易扩展的方式获得所需的服务。这种服务可以是 IT 和软件、互联网相关的，也可以是任意其他的服务。

云计算的目标是：用户可以使用连接网络的简单终端，比如笔记本式计算机，甚至是一部手机，通过 Internet 得到所需要的服务，甚至包括超级计算。换句话说，云计算是一种利用了 Internet 上的强大的数据存储和处理能力，将复杂的运算从用户终端迁移到云中进行的计算模式。

现有的云计算主要具有以下特点[8]：

(1) 大规模基础设施。云计算平台的底层基础硬件拥有相当大的规模，Google 的云计算平台拥有超过 100 万台的服务器，Amazon、Microsoft、Yahoo 等云服务平台的云也均拥有几十万台服务器。私有云一般拥有成百上千台服务器。“云”为用户带来了前所未有的超强计算能力。

(2) 基于虚拟化技术。云计算为用户提供的资源都是经过虚拟化的资源，用户可以在任何位置使用终端获取所需的服务。用户获得的资源来源于“云”，而并不是固定的有形的实体，用户的应用运行在“云”中，用户不需要了解其应用运行的具体位置。

(3) 高可靠性。数据的多副本容错、计算结点同构互换等策略的使用，使得云计算比本地计算具有更高的可靠性。

(4) 普适性。云计算所提供的服务并不针对某一具体的应用，在云计算的平台中，用户可以根据自己的需要构造出不同的应用，同一个云计算平台可以运行不同用户的不同应用。

(5) 易扩展性。“云”的规模可以根据需要进行动态扩展，以满足云计算应用和用户数量的动态变化。

(6) 按需索取的服务形式。“云”中的计算资源作为一种商品，可以像传统的水电煤气那样按需要购买。由云计算服务提供商根据用户对服务的使用量进行计费。

(7) 低成本。容错技术的使用使得云计算的硬件基础设施可以建立在大规模廉价的服务器集群上。采用价格低廉的服务器构成“云”中的结点，对于云计算服务提供商而言，大大降低了底层硬件的构架成本；对于用户而言，能够用低廉的价格完成很多需要性能强大但价格昂贵的应用服务，而且不必考虑软硬件的维护。

1.8　网格计算与云计算的区别

云计算与网格计算从定义上来看，二者都试图将各种 IT 资源看成一个虚拟的资源池，然后向外提供相应的服务。云计算试图让“用户透明地使用资源”，而网格计算当初的口号就是让“使用 IT 资源像使用水电一样简单”。

根据维基百科所提供的定义，云计算是一种宽泛的概念，它允许用户通过互联网访问各种基于 IT 资源的服务，这种服务允许用户无须了解底层 IT 基础设施架构就能够享受到作为服务的“IT 相关资源”。

而网格的内涵包括两个方面：一方面是所谓的效用计算或随需计算，在这一点上面，网格计算跟云计算是非常相似的，都是通过一个资源池或者分布式的计算资源来提供在线的计算或存储等服务；另一方面就是所谓的“虚拟超级计算机”，以松耦合的方式将大量的计算资源，连接在一起提供单个计算资源所无法完成的超级计算能力，这也是狭义上的网格计算跟云计算概念上最大的差别。

网格计算与云计算的区分如下：

1. 目标不同

一般来说，谈到网格计算大家都会想到当年风靡一时的搜寻外星人项目，也就是说通过在本机安装一个屏幕保护软件，就能够利用每个人的 PC 闲暇时的计算能力来参与搜寻外星人的计算。这也说明了网格的目标，就是要尽可能地利用各种资源。它通过特定的网格软件，将一个庞大的项目分解为无数个相互独立的、不太相关的子任务，然后交由各个计算结点进行计算。即便某个结点出现问题，没有能够及时返回结果，也不影响整个项目的进程，甚至即便某一个计算结点突然崩溃，其所承担的计算任务也能够被任务调度系统分配给其他结点继续完成。应该说，从这一点来说，作业调度是网格计算的核心价值。

谈到云计算的时候，就能够立刻想到通过互联网将数据中心的各种资源打包成服务向外提供。一般来说，尽管云计算也像网格计算一样将所有的资源构筑成一个庞大的资源池，但是云计算向外提供的某个资源，是为了完成某个特定的任务。比如说某个用户需要从资源池中申请一定量的资源来部署其应用，但不会将自己的任务提交给整个网格来完成。从这一点来看，网格的构建大多为完成某一个特定的任务需要，这也是会有生物网格、地理网格、国家教育网格等各种不同的网格项目出现的原因。而云计算一般来说都是为了通用应用而设计的，没有专门的以某种应用命名的网格。

2. 分配资源方式的不同

对于网格计算来说，其资源已经被池化，在外界看来就是一个巨大的资源池。对于要提交特定任务的用户来说，他并不知道自己的任务将会在哪些网格的物理结点上运行。他只是按照特定的格式，将作业任务提交给网格系统，然后等待网格返回结果。网格作业调度系统自动找寻与该任务相匹配的资源，然后寻找出空闲的物理结点，将任务分配过去直至完成。虽然网格能够实现跨物理机进行并行作业处理，但是需要用户先将并行算法写好，并且通过调度系统将作业分解到各个不同的物理结点进行，这个过程相对比较复杂，这也是很多网格计算被建设用来完成特定需求的原因。

云计算是通过虚拟化将物理机的资源进行切割的，从这个角度来实现资源的随需分配和自动增长，并且其资源的自动分配和增减不能超越物理结点本身的物理上限。尽管从控制端来看，云计算也将所有的IT资源看成是一个资源池，但是不同芯片的物理机会被归类到不同的资源池中。比如说为呼应某一个应用的请求，而给其分配一颗x86 CPU、Power CPU或者Itanium CPU，分配内存和硬盘空间，再给其安装Linux系统及相关的应用，但是不能同时分配一颗x86 CPU和一颗其他的CPU以构成一个异构的环境。而且，如果结点中的物理机最高CPU数量是4颗的话，那么即使由10台这样的结点构成一个40颗CPU的资源池，也不能为某一个应用分配8颗CPU的虚拟结点。

从这种角度来说，Amazon在2006年所推出的EC2(Elastic Compute Cloud，弹性计算云)项目的确算得上是云计算项目，只不过那个时候云计算概念未兴起，而网格计算的概念方兴未艾，Amazon在那个时候依然用网格的概念向外推销该项目。

网格计算与云计算二者的相同点在于，无论是用户还是企业开发者，都能够通过Internet来获得数据或者进行计算。尽管本地资源有限，但是能够通过网络进行复杂的运算，其数据的计算过程对于用户来说就像互联网网络对于本地网络用户一样，正如大家所了解的网络云，后端的实现是透明的。它们是殊途同归的，都能够被看成是分布式计算所衍生出来的概念，都是为了让IT资源能够对用户透明，为了让IT资源能够达到更好的使用率。从提高资源利用率的角度出发，逐渐诞生了Web服务的概念，然后网络公司通过部署数以万计的服务器构成庞大的计算资源，进而提供此前无法完成的新服务。企业或者个人能够通过Internet利用那些大公司所释放出来的计算资源，进行应用部署或者向外提供服务。这就是从网格计算到云计算演变的历史过程。

1.9 本书研究的主要问题

本书针对目前最新的三种计算模式——对等计算模式、网格计算模式以及云计算模式的关键技术展开研究，下面将对主要的研究重点进行详细介绍。

1.9.1 对等计算

过去10年见证了Internet涌现出的大量分布式应用，其中最流行的应用之一是使用P2P系统实现文件共享。1999年1月，Shawn Fanning离开美国的西北大学，开发了软件Napster[9]。Napster是公共可用的第一个用于音乐共享的系统，并且取得了极大的成功。在它诞生后不久，世界上就有几百万的用户使用它。然而，好景不长，美国唱片工业协会(RIAA)发起了对Napster的起诉，认为它在非法共享MP3音乐文件方面起了推波助澜的作用，因此违反了版权法。Napster因此被司法局在2001年3月关闭。但Napster的结束并不意味着P2P文件共享的结束。相反，以Gnutella为主力的分布式P2P系统仍然在继续发展并不断壮大。新系统如KaZaA[10]、LimeWire[11]、Morpheus[12]不断出现，P2P用户数量持续快速增长。2000年夏，KaZaA软件已被超过1亿用户下载。此外，系统不再局限于共享音频、视频、软件和其他格式文件，它已经超越了文件共享的界限，出现了如SETI@Home[13]等利用系统参与者的空闲处理能力等新型应用。今天，P2P系统已经在数据存储[14~16]、组通信[17,18]、消息系统[19]等多方面得到了应用。

P2P 系统开辟了 Internet 应用的一个新时代，各种新型应用层出不穷，如雨后春笋般吸引了千百万的用户使用。反过来，这种情况也促使对 P2P 系统的性能有了更高的要求。研究 P2P 系统的关键技术，改善 P2P 系统的性能，成为一个重要的课题。

P2P 系统是一个迅速发展的研究领域。P2P 系统的应用已从传统的文件共享领域逐步扩展到更广泛的广域分布计算领域，因而需要 P2P 系统提供确定性定位与低查询开销等关键特性。基于分布式哈希表(Distributed Hash Table，DHT)的 P2P 系统在广域网支持海量的数据一致性分布，并提供低跳步的路由精确定位，以及具有低查询开销和高容错自组织等优良性能，已经成为学术界研究的热点。

分布式哈希表技术密切相关于 P2P 系统的设计，主要是拓扑、路由和查询这三个紧密相关的关键设计技术，并由此深刻影响着 P2P 系统的"资源定位和查找"这一个系统应用的核心问题。首先，分布式哈希表技术引入是对传统 P2P 系统的拓扑结构的一种根本变革，将拓扑结构由一种随机的、无序的非结构拓扑转变为确定的、有序的结构化拓扑；其次，分布式哈希表技术引入导致了路由技术的根本不同，由传统 P2P 系统的基于无向性的泛洪路由技术而转变为基于精确定位导向的单播路由技术；最后，由于分布式哈希表技术对于拓扑、路由的变革影响，也导致了查询由传统 P2P 系统的自由式的随机查询且查询结果具不确定性转变为严格控制的查询且查询结果具确定性。这些变革特性，使得 P2P 系统在广域网对诸如网络存储、组通信、名字服务、信息搜索等应用提供了极具竞争力的优良性能，从而大大超越了传统的文件共享的应用界限，给 P2P 系统带来新的发展和机遇。

然而，分布式哈希表技术的引入在带来其先进性的变革影响的同时，也带来了新的挑战性问题。第一，由于拓扑是一种结构化的拓扑，对比非结构化的拓扑，其维护开销显著加大。特别是在大规模和动荡的网络环境下，维护开销相当可观。第二，尽管路由采用了精确的定向单播技术，但是减少其路由跳数仍然是个重要的问题。在保证高度的分散性前提下，如何尽可能减少其路由跳数是研究人员特别关注的问题，这个问题又称路由的状态与效率折中问题。第三，基于分布式哈希表的查询是一种单关键字的精确匹配，尽管相对于非结构化系统，它可使得系统资源能被确定性地查询到，但它也极大地限制了查询的应用范围。这些问题对于 P2P 系统的可用性和效率是重要而又关键的。本书研究了 P2P 系统中应用分布式哈希表技术所存在的极富挑战性的问题，在保障分布式哈希表技术带来的优良性能的前提下，在一定程度上突破了 DHT 技术应用的局限性，初步解决了上述的挑战性问题，取得了有价值的进展。本书的主要的研究成果如下：

(1) 提出了在动态网络环境下自适应的拓扑调整模型(Session Heterogeneity Topology，SHT)，能够有效地控制 DHT 拓扑维护开销并提高了系统资源的可用性，较好解决了 DHT 拓扑维护的高开销问题。

SHT 利用结点的会话异构特性将 DHT 重构，将 DHT 的拓扑构造变成仅由稳定结点组成，从而大大降低了拓扑适应动态环境的调整开销。研究揭示，会话时间短的结点是拓扑调整的主要扰动因子，因此建议拓扑设计时将它们聚簇于稳定结点，并由稳定结点代理它们的请求，从而在保证它们能够正常使用系统的同时又将它们的扰动限制于稳定结点的动荡。理论分析和实验表明一个合理的聚簇大小的存在，使得拓扑达到近似最优化，此时减少的开销达到近似最小。

(2) 提出了基于小世界理论的概率缓存链技术，能够在高度分散的低状态路由表下达

到较高的路由效率，较好解决了 DHT 路由的低效率问题。

研究利用了 DHT 路由的贪婪特性及查询的反馈机制，构造了概率缓存的路由快捷链，由此在路由表链间形成了一个小世界，使得系统的路由性能得到全局性的优化。设计充分考虑了静态与动态下的收敛问题，并设计主动查询机制以适应动态环境并加速收敛，使系统性能得到保障。由于设计采用的是查询反馈机制以及缓存技术，具有较强的现实意义。特别是对于当前的一大类采用低状态 DHT 路由表的具有高度松散结构的 P2P 系统，能够在轻微的修改下达到路由效率可观的提高。

(3) 提出了基于向量空间模型(VSM)的相似文档搜索方法和技术，使得 DHT 查询能支持多关键字查询和相似文档搜索，从而突破了 DHT 查询的单关键字的精确匹配约束，使 DHT 查询的应用范围大为扩展，较好解决了 DHT 查询的应用局限性问题。

研究在两方面对 DHT 查询进行了改进。一方面，目前基于 DHT 的多关键字查询技术，通常采用单关键字的组合查询的模式，带来较多的网络开销。通过 VSM 技术用多个关键字形成向量空间模型 VSM 中的向量，并使得查询相关于此向量，从而能够支持多关键字查询并去除了单关键字组合查询所需的大量网络开销。另一方面，当前 DHT 查询的局限性本质是由于哈希转换使得查询失去了语义性支持，仅能够支持精确匹配，而利用 VSM 技术进行文档标识的哈希空间重映射，使得 DHT 文档标识具有相似性的语义，从而支持相似性文档搜索。

随着 P2P 信息资源继续快速的增长，信息无序和信息过载即将成为一个越来越迫切需要解决的问题。于是人们提出了 P2P 挖掘技术。简单地说，P2P 挖掘类似于 Web 挖掘，就是使用数据挖掘技术自动地从对等结点的文件和服务中发现、提取信息和知识的技术，但 P2P 挖掘特殊性在于它是一个分布式的挖掘。它是一个交叉的研究领域，涉及分布式计算、传统的数据挖掘，以及文本、图像、声音、视频等方面的信息检索(Information Retrieval，IR)、信息抽取(Information Extraction，IE)、人工智能、模式识别，特别是机器学习和自然语言处理(Nature Information Process，NLP)等领域。

1.9.2 网格计算

随着计算机技术和网络技术的发展，以及日益增长的计算力的需求，诞生了网格计算。构建一个网格系统需要研究信息服务、数据管理、安全机制等技术。资源管理对高效合理利用计算资源起着十分重要的作用。网格资源具备动态性、异构性和自治性的特征，需要对网格资源管理和调度的关键技术做相关研究。

网格资源的动态性、异构性和自治性使得网格资源提供者和使用者具有对等网络的特点，因此有必要将对等网络方法引入到网格资源管理中。为此，本书基于对等网络技术提出了一系列网格资源管理和调度策略。

根据网格的发展历史以及研究工作的针对性，网格可以划分为两大类：一类是专用网格，例如计算网格；另一类是通用网格，即在业界参与之下基于 Web 服务和 OGSA 的网格系统。本书关于网格计算的研究工作适用于这两类网格的资源管理和调度。

本书的网格计算部分首先介绍了网格资源管理和调度策略的一些基本概念和主要的研究内容，然后结合网格自身特点和发展趋势，对其中的几个关键问题进行了深入的研究，包括网格系统的体系结构，网格资源信息的表示方法、资源管理和调度算法，以及负载均衡。

为了验证本书提出的模型和算法的有效性，进行了大量的实验，实验结果证明了本书提出的模型和算法的有效性。

本书关于网格计算的研究重点如下：

(1) 在分析了网格自身特点和发展趋势的基础上，本书将对等网络方法引入网格的资源管理和调度中，结合对等网络的完全分布式的资源管理方式的优点，设计了基于超级结点对等网络的网格资源管理体系结构。这种集中式和分布式的混合设计结构能够解决现有网格系统采用的集中式管理的容易引起的单点失效、性能瓶颈等问题，从而可以更好地描述网格资源的动态性、自治性等特点，使网格系统具有更强的鲁棒性(即稳健性)和自适应性，并且有利于制定优化网格资源管理和调度的策略、算法。进一步根据网格资源提供者的 IP 层信息生成含有路由信息的叠加网络(Overlay Network)拓扑，并且使用有向图表示该拓扑结构。这种使用有向图进行网格拓扑结构表示的方式在能够准确描述网格资源提供者的计算能力的同时，还能够弥补其他现有的资源信息表示模型的 Overlay 层路由信息不能精确反映 IP 层路由情况的不足，同时这种简单的描述方式有利于网格资源调度器发掘网格资源提供者和网格任务之间的对应关系。

(2) 提出了基于树匹配的 nTreeMatch 算法。该算法结合 DAG 图(Directed Acyclic Graph，有向无环图)的任务表示形式，通过树形数据结构匹配的方法解决了网格资源和网格任务间的映射问题。同时，该算法充分利用 Overlay 拓扑中结点的路由信息，以轻量附加开销来有效减少 Overlay 层上的路由跳数，使得 Overlay 层上的路由跳数尽量接近 IP 层上的路由跳数，降低 RDP(Relative Delay Penalty，相对平均延迟开销)。理论和模拟实验表明在大规模的网格系统中，算法在进行资源调度时可以获得较高的路由效率，为路由的状态与效率折中问题提供了一个可行的解决方案。该算法尤其适用于为特定的科学应用而设计的专用计算网格的资源调度。

(3) 针对基于 Web Service 的通用网格系统的资源调度，本书提出了基于资源发现的 GChord 算法。考虑到网格的动态性特征，GChord 算法采用服务发现的方式解决资源调度问题，将资源需求按照 Chord 路由协议在网格中转发，改变了传统的集中式调度方法采取的信息收集方式，能够实时反映网格结点的工作负载状态，有效解决由于信息过时、数据不一致而引起的任务再调度问题。实验证明，GChord 算法可以实现网格系统的实时资源调度，并且使得网格系统保持良好的负载均衡状态。

(4) 为解决网格资源调度中动态负载均衡的挑战，在研究了多代理技术和网格计算相互融合的发展趋势的基础上，本书提出了基于多代理协同计算的 rwAgent 算法。该算法利用多代理技术，通过代理的自治性和智能学习，实现网格资源的分散调度，同时可以获得很好的负载均衡效果。严格的数学建模和理论分析证明，rwAgent 算法可以实现资源调度过程中网格系统的全局负载均衡，实验结果证明了算法的有效性和优越性。

1.9.3　云计算

1. 云计算技术的特征和优势

云计算通过虚拟化方式提供动态资源池，使得资源的可用性更加高效。与传统的计算模式相比，现阶段的云计算技术具有以下特征[20]：

(1) 云计算的底层硬件基础架构是由大量的廉价服务器集群(甚至是 x86)构成的。在

传统的计算模式中，要想获得性能强大的计算能力，那就意味着昂贵的价格。而云计算使用虚拟化技术手段，利用廉价的服务器，将其资源整合，从而达到既性能强大，又价格低廉的目的。

（2）云计算的底层结构与应用程序协作开发，使得资源利用最大化。传统计算模式中的应用程序必须在完整的软件（操作系统）和硬件基础上运行，也就是说应用要建立在完整的底层硬件之上。而云计算采用了底层硬件架构与上层应用程序协同设计的方法，更好地利用了资源。

（3）云计算通过软件的方法利用服务器集群中失效结点产生的冗余，获得资源优化利用。廉价服务器之间的结点失效问题不可避免，它不仅会影响系统性能，还会造成资源的利用效率低下。在设计软件时考虑到结点之间的容错问题，有效利用冗余结点，不仅可以提高资源的利用率，还可以提高整个系统的性能。

从以上云计算具有的特征来看，云计算具有很多传统计算模式所不具备的优点。从用户角度看，云计算的特点如下：

（1）云计算平台通过专业人员的管理，具有可靠性高的存储技术以及权限策略，以此为用户提供了安全可靠的数据存储中心。“云”用户不用再考虑数据丢失和病毒入侵等问题，可以放心地使用云计算的服务。

（2）要求低，使用方便。云计算对用户端的设备要求低，简单的终端设备如PC机、笔记本式计算机，甚至是手机等无线通信设备，都可以随时随地地通过Internet接入到“云”中，在浏览器中直接编辑存储在“云”中的文档。

（3）云计算可以轻松实现不同设备间的数据与应用共享。所有电子设备只需要连接到Internet，就可以对数据与应用进行操作。

（4）云计算使得网络可以为用户提供更多服务。它为数据的存储以及管理提供了更多的空间，也为完成各类应用提供了强大的计算能力。云计算可以轻而易举地做到PC机或其他电子设备不可能做到的事情，比如云计算可以为用户提供几乎无限的存储空间和超级的计算能力，这是因为云计算的底层架构是由成千数万甚至更多的服务器组成的超大规模的集群。所以说云计算的潜力是无穷的。随着云计算的进一步发展与完善，用户充分相信“云”的时候，整个IT行业会发生翻天覆地的变化。

云计算还具有分布式计算特有的优势：与集中式系统相比，具有更高的性价比，也就是花更少的钱得到更高效的计算。在现实生活中，大多数应用本身就是呈分布式的，比如在工业或企业中，管理部门和应用现场基本上都不在同一个地方；分布式的高可靠性、可扩展性、冗余技术使得现代分布式系统具有高度容错机制，购买一台性能高的大型机的费用远远高于增加几台PC机的费用；分布式计算具有灵活性，可以兼容不同硬件厂商的产品，甚至兼容低配置设备而获得高性能的计算。

云计算在存储方面的优势是：用户不必为文件存储投入任何前期的费用，基础设施服务提供商会维护用户文件服务器的安全和更新问题。

2. 云计算的关键技术

1）编程模式

前面提到云计算可以使得用户或使用云计算服务进行开发的程序人员，从复杂而烦琐的软硬件的维护与升级中解放出来，使他们可以将精力放在只与其业务有关的工作中，所以云

计算所采用的编程模式应该尽量使后台复杂的并行执行和任务调度等对用户和编程人员透明。为了方便用户利用云中的资源，云计算的编程模式应该尽量简单易学。MapReduce[21] 是 Google 提出的适用于云计算的一种新兴的编程模式。目前，几乎各个云计算服务提供商所使用的编程模式都是以 MapReduce 为基础的。MapReduce 虽然是针对云计算提出的，但是在并行处理以及多核计算上也都具有很好的表现。但是目前的 MapReduce 仅用于编写处理数据为主的、并行化程度高的程序。这是因为 MapReduce 还不能够对有相互联系的计算数据进行处理，这也是 MapReduce 接下来要改进的地方。

2）数据存储管理

云计算的数据存储和管理采用的是分布式方式，这样做有利于保证位于云中的数据具有更高的可用性及可靠性。冗余存储是云计算中采用的一种安全存储数据的技术，它可以通过存储系统的高容错性进而提高数据存储的可靠性。云计算采用分布式的方式对数据进行存储，这是因为云计算平台需要同时并行地为多个用户提供服务，而分布式的存储方式可以满足云计算的多用户的特点，保证云计算存储的高吞吐率。

目前，成熟的云计算数据存储技术主要如下：

（1）Google 的 GFS[22]（Google File System，一种非开源的存储体系）：是 Google 提出的一种可扩展的分布式文件系统。为了减少复杂性，GFS 并未遵守 POSIX[23]（Portable Operating System Interface of UNIX）标准。GFS 虽然是分布式文件系统，但并未采用纯分布式的架构，而是采用了中央控制管理服务器。一个 GFS 系统是由一个 Master 和一个大规模的 Chunk server 组成，其中 Master 负责维护和控制文件系统中所有的元数据、名字空间、文件访问控制信息、文件映射信息及数据块位置信息等；Chunk Server 负责数据块的存储，在云计算中，出于数据的安全可靠性考虑，每一个数据块都会被复制到多个 Chunk Server 中。

（2）Hadoop 团队开发的 HDFS[24]（Hadoop Distributed File System，Hadoop 分布式文件系统）：是基于分布式的一种文件系统。因为它的高容错性，所以一般用来部署在廉价的计算机硬件设备上。它的高传输率使得拥有大规模数据集的应用程序能够高效快速地访问到其需要的数据。HDFS 被众多 IT 企业如 Yahoo、Intel、Alibaba 等作为数据存储的技术。

云计算系统需要在处理来自于大量用户的数据的同时，保证服务的高效性。所以云计算的数据管理技术必须能够高效可靠地对海量的用户信息进行处理。云计算的数据管理模式采用的是读操作优先的模式。这是因为对存储在“云”中的数据进行读操作的频率远远超出了其他操作的频率。读操作优先模式等同于数据库中的列存储数据管理模式，它首先将表按列进行划分，然后再进行存储。Google 的 BigTable[25] 是一种典型的数据管理技术。

3）虚拟化技术

云计算中的核心技术就是虚拟化，可以说是虚拟化为我们带来了“云”，同时也是云计算区别于传统计算模式的重要特点[26]。虚拟机是一种可以完全模拟硬件执行的特殊软件，它运行在完全隔离的环境中，所以可以将操作系统运行其中，这样做可以对运行环境进行有效的保存。采用虚拟化可以将应用程序的整个执行环境以打包的形式转到云计算平台中的其他结点处，实现程序的执行环境与物理环境的隔离，使得应用程序的环境变得易于实现。

使用虚拟化技术可以将计算机硬件设备进行逻辑上的扩大，大大简化了软件的多次配置过程。例如，可以通过虚拟化技术将一个CPU模拟成多个并行的CPU，通过虚拟化可以将多个操作系统同时运行在一个计算机平台上，而且在整个系统中，应用程序的运行是相互独立的，不会互相影响。这样使得计算机的效率得到了很大的提高。

虚拟化技术不同于传统的多任务或超线程技术，后两者指的是多个任务或程序运行在单个操作系统上，而虚拟化指的是多个操作系统同时运行，而每一个操作系统在同一时间都运行多个应用程序，每个操作系统又都运行在一个虚拟的主机之上。传统的超线程技术只是实现了通过单CPU对双CPU的模拟，来进行程序的运行性能的平衡，而且模拟的双CPU只能相互协同工作。云计算使用虚拟化技术可以得到以下好处：

(1) 虚拟化技术使得云计算动态找到计算所需要的资源，然后将其定位到合适的物理平台之上，在整个过程中，运行在虚拟机中的程序不需要退出。

(2) 虚拟化技术使得位于云中的物理主机的资源利用率大大提高，虚拟化将所有闲置计算资源的计算结点整合到同一个物理结点之上，这样能够节约维持多个物理结点所需要的成本。

(3) 虚拟机的动态迁移有利于系统的性能做到负载平衡，而且迁移的虚拟机中拥有整个应用程序的运行环境。

(4) 虚拟化技术让云计算平台的部署变得灵活，云计算服务提供商可以将一个虚拟机作为资源提供给用户，也可以根据用户的需要提供相应的资源。Amazon 的 EC2 就类似于提供给用户的一个完整的虚拟机。Xen、VMware 以及开放源代码的 Linux KVM 是几种比较成熟的虚拟化技术。

1.10 本书的层次结构

本书共分为10章，层次结构如图1-1所示。

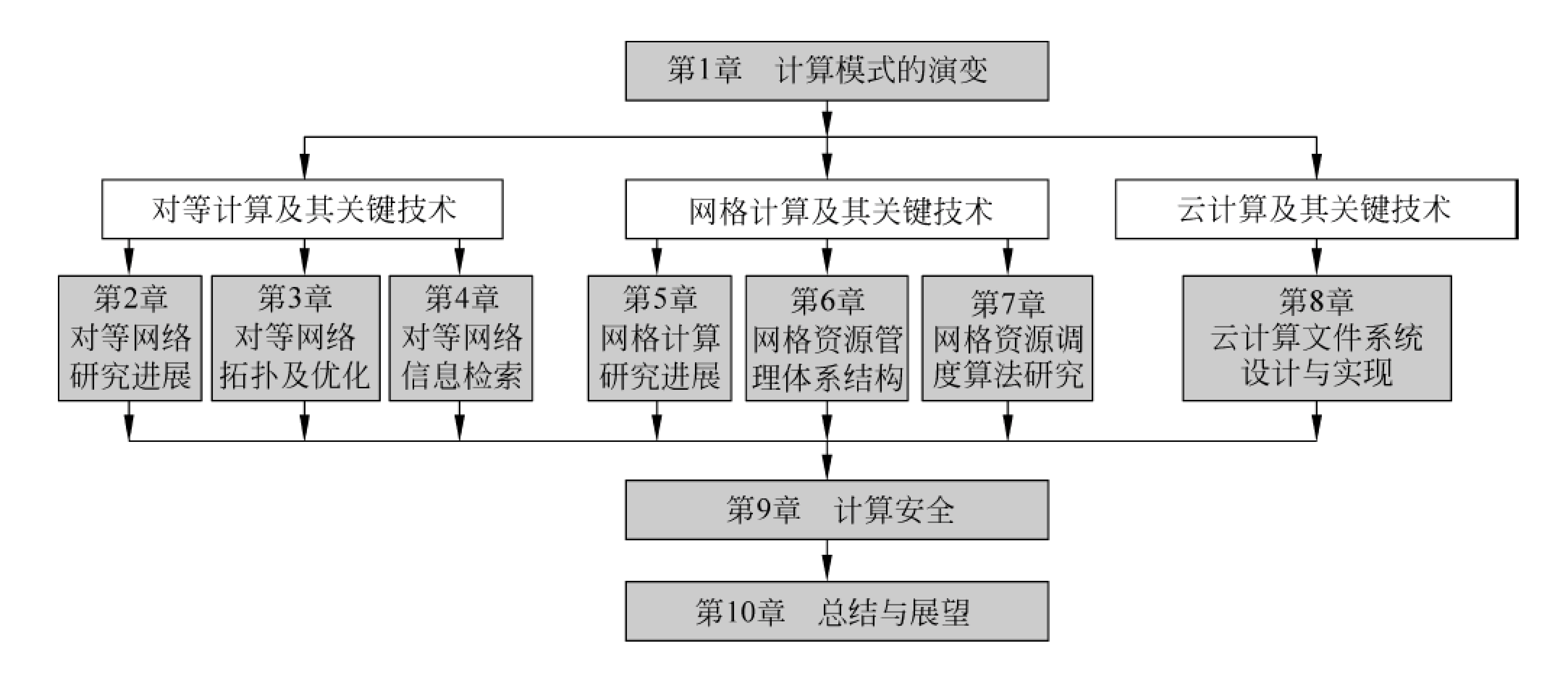


图1-1 本书的层次结构图

参考文献

[1] Foster I, Geisler J, Nickless W, Smith W, Software Infrastructure for the I-Way High Performance Distributed Computing Experiment, Proc. 5th IEEE Symposium on High Performance Distributed Computing. 1997, 562-571.

[2] Foster I, Kesselman C. Globus: A Meta Computing Infrastructure Toolkit, International Journal of Supercomputer Applications, 1997, 11(2): 115-128.

[3] Foster I, Kesselman C. The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann, 1998, 7.

[4] Foster I, Kesselman C, Tuecke S. The Anatomy of the Grid: Enabling Scalable Virtual Organizations, Int. Journal of Supercomputer Applications High Performance Computing, 2001. http://www. globus. og/research/papers/ogsa. pdf.

[5] Ian Foster. What is the Grid? A Three Point CheckList.

[6] Ian Foster. A talk on Grid Computing: Concepts, Applications, Technologies. http://www-fp. mcs. anl. gov/～foster/Talks/WWWGridsMay2002. ppt.

[7] Globus Toolkits 4. 0. http://www. globus. org.

[8] WIKI. http://zh. wikipedi. org/wiki/Cloud Computing, 2010.

[9] Napster. http://www. napster. com.

[10] KaZaA. http://www. kazaa. com.

[11] Limewire. http://www. limewire. com.

[12] Morpheus. http://www. musiccity. com.

[13] SETI@Home. http://setiathome. ssl. berkeley. edu.

[14] John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westley Weimer, Chris Wells, Ben Zhao. OceanStore: An Architecture for Global-Scale Persistent Storage. ACM SIGPLAN Notices, 2000, 11, 31(1): 190-201.

[15] Rowstron A, Druschel P. Storage management caching in PAST, a large-scale, persistent peer-to-peer storage utility. In Proc. of SOSP, ACM, 2001, 10: 188-201.

[16] Bolosky W J, John R Douceur, David Ely, Marvin Theimer. Feasibility of a Serverless Distributed File System Deployed on an Existing Set of Desktop PCs. In Proceedings of the ACM SIGMETRICS International Conference on Measurement Modeling of Computer Systems, Santa Clara, CA, USA, 2000, 6: 34-43.

[17] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Antony Rowstron. SCRIBE: A large-scale decentralized application-level multicast infrastructure. IEEE Journal on Selected Areas in Communications (JSAC) (Special issue on Network Support for Multicast Communications), 2002, 10, 20(8): 1489-1499.

[18] Rowstron A, Kermarrec A M, Druschel P, Castro M. SCRIBE: The design of a large-scale event notication infrastructure. In Proceedings of the Third International Workshop on Networked Group Communication (NGC), volume 2233 of Lecture Notes in Computer Science, UCL, London, UK, 2001, 11: 30-33.

[19] ThreeDegrees. http://www. threedegrees. com.

[20] 曾诚. 云计算的栈模型研究[J]. 微电子学与计算机. 2009, 8.

[21] Dean J, Ghemawat S. MapRecluce: Simplified Data Processing on Large Cluster [C]. Sixth Symposium on Operating System Design Implementation. SanFranciseo, CA. 2004: 149-167.

[22] Ghemawat S,Gobioff H,Shun-Tak L.(2003). The Google file system,ACM. SOSP'03.
[23] IEEE POSIX. http://posixcertified.ieee.org.
[24] HDFS. http://wiki.apache.org/hadoop/HDFS,2010.
[25] Fay C,Jeffrey D,Sanjay G. BigTable: A Distributed Storage System for Structured Data. http://labs.google.com/papers/bigtable.html,2006.
[26] Chen H B, Chen R,et al. Live updating operating system using virtual[c]. In Proceedings of the 2nd International Conference on Virtual Execution Environments. 2006: 33-42.

CHAPTER 2

第2章 对等网络研究进展

随着计算机技术、网络技术、通信技术的发展,计算模式也正面临着一场革命。这场革命正在改变 Internet 的一个基本特性,这就是网络的终端(或称客户端),如桌面电脑、移动电话、PDA 等正要求一个更优越的网络地位,以结束服务器为主导的 Internet。这就需要 P2P 计算。

P2P 计算技术出现的目的就是希望充分利用 Internet 中所蕴含的潜在计算资源。P2P 即对等网络,指分布式系统中的各个结点是逻辑对等的,与目前 Internet 上比较流行的 C/S 计算模型不同的是:P2P 计算模型中不再区别服务器以及客户端,系统中的各个结点之间可以直接进行数据通信而不需要通过中间的服务器。和传统的 C/S 网络相比,对等网络具有扩展性好、容错性好、负载均衡和整体性能高等优点。

对等网络的以下特征使之与传统的计算机系统相区别:

(1) 结点之间通过直接交互共享资源。

(2) 资源分布在各个结点中,而不是集中在一个服务器中进行管理。

(3) 结点具有动态性和自治性。

(4) 纯粹的 P2P 系统没有集中控制机制。系统的各结点运行的 P2P 系统软件功能相同,各结点之间交互对称。

2.1 对等网络概述

2.1.1 对等网络的定义

对等网络正处于不断发展的阶段,因而并不存在一个精确的定义。当前给出的许多定义都是试图反映 P2P 系统发展过程中的某阶段的新特征。下面是文献针对 P2P 系统给出的一些定义。

1. Clay Shirkey

P2P 是一种利用位于 Internet 边缘的各种可用资源(如存储空间、计算能力、媒体内容)的应用。访问这些分散的资源,就意味着要在连接不稳定和

IP 地址不可预见的环境里工作。由于网络上大量的结点工作在 DNS 系统之外,这些分散的资源具有不稳定的连通性和未知的 IP 地址。因此,P2P 结点必须能够独立于 DNS 系统且高度自治[1]。

2. Mike Miler

P2P 是一个网络体系,其中每台计算机有同等能力和责任,有如下 5 个关键特性[2]。

(1) 网络提供结点间实时的数据传输或者消息传递。

(2) 结点既是客户端又是服务器。

(3) 网络的内容是由分布的结点提供。

(4) 结点具有网络控制权和自治权。

(5) 网络允许不总是连接的结点和可能没有永久 IP 地址的结点参与。

3. P2P 工作组

P2P 是通过在系统之间直接交换来共享计算机资源和服务的。这些资源和服务包括信息交换、高速缓存、处理能力、存储空间。P2P 可以整合这些 PC 机上的计算力和网络连接,从而提供企业级的计算平台[3]。

一般地说,P2P 是一个用于资源共享的对等结点群体,其中每个对等结点向群体提供资源的同时,作为回报从中获取所需资源。它的思想是基于世界上的事物是广泛分布且相互联系的,不可能通过一种集中化的方式管理如此庞大的结构。P2P 通过分布于世界各地的 PC 机管理大量的计算能力、存储空间和连接。P2P 中的每个对等结点自治又彼此依赖,所谓自治是指每个对等结点独立决定自己的行为而不受其他机构(如集中式授权机构)的控制,同时每个对等结点又需要相互协作获得信息资源、计算资源。在本书中把每个 peer 称为对等结点,并把 peer 所组成的网络称为叠加网络。

Intel[3]的定义:P2P 是通过系统间的直接交换所达成的计算机资源与信息的共享。这些资源包括信息交换、处理器时钟、缓存和磁盘空间等。P2P 利用了已有的桌面计算能力和网络连接,将这些资源和服务有效组织,通过更强大的集合能力来完成任务。

IBM[1]的定义:P2P 系统由若干互联协作的计算器构成,且至少具有如下特征之一。系统依存于边缘化(非中央式服务器)设备的主动协作,每个成员直接从其他成员而不是从服务器的参与中受益;系统中成员同时扮演服务器与客户机的角色;系统应用的用户能够意识到彼此的存在,构成一个虚拟或实际的群体。P2P 将网络抽象成由主机和主机间的应用层链路组成,把网络计算模式从集中式引向分布式,网络应用的核心从中央服务器向网络边缘的终端设备扩散。

2.1.2 对等网络的发展历史

文件共享是 P2P 最成功的一种应用,这里以文件共享为例来说明 P2P 的发展。在文件共享领域为什么会引入 P2P 呢?最开始的文件共享采用的是 FTP 协议,但 FTP 协议存在发现资源困难的缺点,尤其是当提供资源共享的 FTP 站点规模很大时。其主要原因是网络中的结点标识(IP 地址)和结点拥有的文件两者间不存在必然的对应关系,即结点所拥有的文件和其 IP 地址之间相互独立(独立事件)。这就造成了不可能知道某个 IP 地址的结点上是否有某个文件,也不可能知道某个想要的文件会在哪个 IP 的结点上。这使得 FTP 搜索

引擎的出现成为必然。FTP 搜索引擎负责将 FTP 站点中的共享资源搜索出来,放到本地数据库中,以便用户查找。但这是一种由 FTP 搜索引擎主动进行内容收集,存在搜索信息滞后、内容不全面和单点故障等问题。P2P 中的第一代应用——Napster 采用的是由结点主动注册的机制,其仍然存在单点故障、负载不均衡等缺陷;Gnutella 系统采用泛洪方式来传递查询消息,这样会产生大量的查询负载。第一代的 P2P 应用系统尽管实现简单,但缺点也是比较明显的。为了解决 P2P 系统 Napster 中目录服务器的单点故障的问题,一般的思路是采用多个服务器共同存储共享文件的索引,P2P 将这个思想进一步推广,将整个共享文件的索引放到整个网络结点上去索引,随之而来的问题就是如何确定资源拥有结点和发布结点之间的对应关系,因此第二代 P2P 系统引入了 Overlay 层,建立了物理网络和 Overlay 层之间的映射关系(由哈希函数确定),这使得资源查找突破了单点故障、负载不均衡等缺点。第二代 P2P 系统以 Chord[4]、Tapestry[5]、Pastry[6]和 CAN[7]等为代表。

2.1.3　对等网络的分类

P2P 具有多种分类标准。这里仅选取分散度和网络结构[8]为分类标准,因为这种分类方法是恰当地反映了 P2P 系统的本质特征。

分散度表示 P2P 系统中结点在进行相关信息搜索时依赖目录服务器的程度。它有三种情况:完全分散、部分分散、混合分散。在完全分散的情况下,所有结点都是平等的,没有任何一个会比其他一个更重要,不存在目录服务器;在部分分散的情况下,一些所谓的超级结点充当了部分目录服务的功能以改善系统性能;在混合分散的情况下,整个系统依赖于一个或者非常少的不可替代的结点提供集中式的目录服务,而系统内其余的结点,具有彼此等同的功能。

网络结构是从拓扑透视的角度来看系统,它表示 P2P 系统的覆盖网络结构是受某种机制约束还是完全动态的,前者为结构化,后者为非结构化。非结构化 P2P 系统中对等结点连接任意其他对等结点构成对等网络,数据放置与网络拓扑无关。系统中每一个结点只负责管理自己的数据文件,从而它的网络拓扑是一个随机连接图,不能保证一定可以找到目标结点。结构化 P2P 系统的网络有某种预定的结构,如环或者网格,系统中每一个数据文件所放置的位置由特定的协议确定,并提供了文件标识 ID 和文件存储位置之间的映射关系,从而保证具有有效路由,并能够保证最终找到目标结点。

结构化系统采用分布式哈希表(Distributed Hash Table,DHT)技术构造,其本质是完全分散构造。因此,仅需要对非结构化系统区分分散度情况。在当前的系统中,Chord、CAN、Pastry 和 Tapestry 是结构化 P2P 系统。而非结构化根据分散度区分为以下三种系统:混合分散的非结构化 P2P 系统,如 Napster;完全分散的非结构化 P2P 系统,如 Gnutella、Freenet;部分分散的非结构化 P2P 系统,如 Kazza Morpheus。根据此标准,P2P 系统的划分如图 2-1 所示。

在网络和分布领域中,P2P 是一个相对新的名词。从 1999 年 Napster 出现及 2000 年 P2P 成为研究热点[9]以来,P2P 系统得到迅速发展。这里按时间出现的先后序,将 P2P 系统的发展大致分为二代。每一代都有其显著的特征:第一代网络结构为非结构化,第二代网络结构为结构化;第一代特点是采用泛洪的不确定查询,第二代特点是采用基于分布式哈希表的确定性查询。在此,仅给出一个轮廓般的介绍,以刻画出其发展的轨迹。

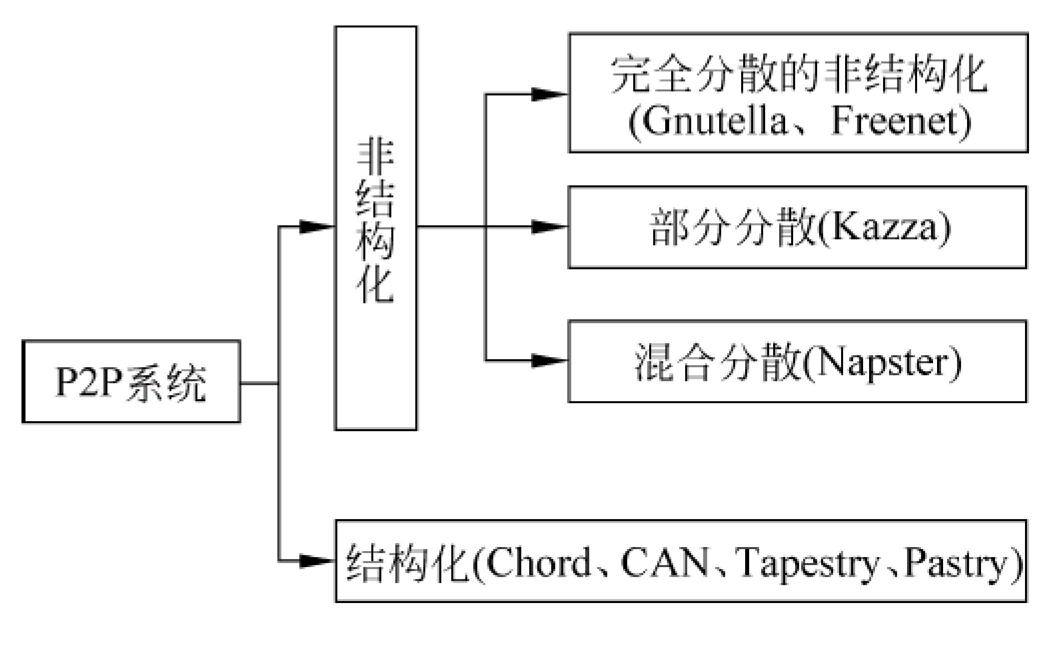


图 2-1　P2P 系统分类

1. 第一代 P2P 系统

第一代 P2P 系统是以非结构化为特征。它是伴随文件共享应用 Napster[10] 而开始的。Napster 的主要贡献是介绍了一种新型的网络体系，这种体系不同于传统的客户端/服务器架构，而是让计算机之间借助一个中心服务器可以自由通信交流。在某种程度上，计算机既是资源的消费者又是资源的提供者，因而对等结点更适合表示系统内的参与者的对等情况。为了在共享空间定位文件，Napster 的解决途径是提供一个中心目录服务器。因而，Napster 系统由两种服务组成：存储服务和目录服务。存储服务是分布的，采用了对等风格，而目录服务是集中式的，因此它是一种混合分散构造。但 Napster 仅短暂存在，它的体系结构中，集中式的目录服务是个关键性问题。首先，它存在单点故障问题；其次，尽管目录服务提供了定位的低开销，但是目录服务器上的负载也是随参与者数目线性增长，从而使得它是不可扩展的。

Napster 集中目录服务器瓶颈导致 P2P 新系统设计关注于如何减少这种集中目录服务，即趋向更高的分散度。Gnutella[11] 和 Freenet[12] 是新的系统代表，它们是完全分散的非结构化系统。这些系统中新的参与者必须知道一个已经参与系统的成员，然后使用一种 Flooding 算法去得到关于别的参与者的情况，以建立邻居关系表。而对于一个给定的查询，也是采用 Flooding 算法。查询请求结点先将此请求通过 Flooding 算法，关联到它的所有邻居，然后它的邻居再做类似的 Flooding 操作，直至查询发现。为了防止 Flooding 过度，请求包附加上 TTL 限制值。当 TTL 值为 0 时，查询立即终止。

新的系统解决了集中瓶颈问题，然而，可扩展问题却更为严重。这是由于 Flooding 算法带来了高的网络流量，此研究见参考文献[13,14]。而且，由于搜索范围受到限制，不能够确定性地发现存在 Gnutella 网络中的一个数据项或者一个资源文件。

Freenet 采用了一个略好些的方案，它是基于文档路由模型的。它利用哈希技术给数据项唯一标识 ID。当数据项插入系统时，尽可能地插入到具有与它的标识最接近的结点。查询是由数据项的标识所引导定位的。但由于 Freenet 网络的随机特性，发现文档的概率并不高。

一些系统在混合分散和完全分散间做了折中，采用了部分分散机制，如 KaZaA[15]。KaZaA 采用超级结点(Super Peer)技术，允许一些结点充当目录服务从而减少了需要定位数据的 Flooding 消息数。尽管如此，查询仍然是不确定的泛洪方式。

1) Napster

Napster[29] 是一种用于音乐文件交换的对等网络应用软件，但是它本身不是一个纯粹

的对等网络系统。它具有一个中央服务器,里面保存了所有注册过的音乐文件,所有查询工作由中央服务器完成;如果有用户正想在 Napster 上搜索音乐文件,并发送了查询的消息,那么 Napster 服务器就搜索它的索引,这个索引就是由各个对等结点发送来的 Metadata 构造而成的。一旦找到,用户就可以从对应的对等结点上直接下载想要的那个文件。从文件共享的角度来看,Napster 是一个 P2P 系统,然而,从索引方面看,Napster 是一个集中式的系统。目前我国的"PP 点点通"也是一种基于此原理的一个应用软件,同样需要中央服务器来完成注册和查询,受单点失效问题困扰,可扩展性不佳。

2) Gnutella

Gnutella[21]是早期的完全非集中式对等网络文件共享系统之一。系统中每个结点维护一些随机邻居结点的信息,通过连续几次路由转发,该结点的消息可以到达其他结点。在 Gnutella 中,一个结点要查找某个关键字,它就通过向邻居结点广播消息,查找的范围用 TTL 参数来控制,找到关键字的结点就回复它的结果集,为了节省带宽,没有找到关键字的结点就不需要回复了。这种查找方式只限于发起者为中心的某个特定半径范围以内,在超出一定范围后不能进一步扩展,因此查找并不能保证可靠性。这种方式是完全非集中式的,从而有效地避免了单点失效问题。但是它采用了广播消息来进行查找,其可扩展性也不佳。

3) FreeNet

FreeNet[12]是一种分布式信息存储和检索系统,其最大的特点是匿名性。文件的发布者、查询者包括文件的持有者在 Freenet 中都是匿名的。为了实现匿名,文件查找通过带回溯形式的爬山算法来进行,效率不高。Freenet 不能提供可靠的内容查找,由于系统会自动删除一些兴趣度低的文件,会导致系统不能保证一定找到网络中的文档,降低了路由效率。

4) Kazaa

Kazaa[15]系统的应用背景与 Gnutella 类似,是目前 Internet 上用户最多的文件共享系统,目前用户数已经超过 4 000 000。在 Kazaa 系统中,为了避免 Gnutella 系统中泛洪流量太大的问题,采用了超级结点策略来减轻泛洪流量,即将超级结点用作局部文件列表服务器,从而可以减少消息的转发。但和中心目录服务器式 Napster 不同的是,Kazaa 的超级结点之间是 Gnutella 式的完全分布的结构。由于超级结点的数量较少,所以网络流量不至于过大,但这种途径仍然是存在可扩展性不佳的问题。

2. 第二代 P2P 系统

第二代的 P2P 系统是以结构化为特征,它以 Chord[4]、CAN[7]、Pastry[5,7]、Tapestry[7]等为代表,系统是完全分散组织。在这些系统中,一个共有的关键特性是基于分布式哈希表机制。在这些系统中,结点以它的一些唯一性属性如 IP 地址,通过哈希表得到唯一标识 ID。数据项是以键值对<key,value>的方式表示,其中键是对于数据项的索引,而值可以是数据项的定位地址,如 IP 或者 URL。通过哈希运算赋予数据索引键以唯一标识,并将此键对应的键值对插入与此键标识最邻近的结点。注意结点和键的哈希空间是相同的。查询时,通过将查询的键通过哈希表得到唯一标识,并通过此唯一标识找到与之最邻近的结点(此结点存储了数据项所在的地址)。为了支持基于标识 ID 的查找,结点将经过哈希表得到的 ID 空间组织成一个结构化的拓扑,如 Chord 中的环(Circle)、CAN 中的超环(Torus)、Tapestry

的树(Tree)。由于这种结构化拓扑,使得数据查找具有可确定性,即数据只要存在叠加网络上就可以以高概率确定性查找到。

当前两代 P2P 系统并存且都得到相应的发展。它们适应于不同的方面,互为补充。第一代系统在文件共享等并不需要确定性的查询结果的领域会有更广阔的前途。因为非结构化性提供松散的组织特性,可以让参与者有充分自由度,而在最大限度减少对 P2P 系统的影响。第二代系统极大拓展了 P2P 系统的应用领域,使 P2P 系统成为分布计算的一个良好的平台,甚至成为下一代因特网应用的基础[19]。这是由于采用分布式哈希表在完全分布的环境下具有高度确定性和高度容错的特性,因而它能够更好地适应大多数分布式应用的要求。

现有的对等网络可按结点路由表(Finger Table)的结构特性来分类[20],可分为非结构化对等网络、结构化对等网络和混合对等网络。在非结构化对等网络中(如 Gnutella[21])中,对等结点连接任意其他对等结点构成对等网络,数据放置与对等网络拓扑无关。Chord[4]、Tapestry[5]、Pastry[6]和 CAN[7]则属于结构化对等网络,这类网络提供了文件的索引位置和文件存储位置之间的映射关系,从而保证有效路由。混合对等网络是仍然采用 Finger 表,但该表的索引项分为两部分,一部分是按照结构化对等网络来构造,另一部分是按照非结构化索引来构造;或者在对等网络中一部分结点的 Finger 表采用结构化方式来构造,另一部分结点的 Finger 表采用非结构化方式构造。

对等网络 Overlay 层上的关键问题是路由效率问题。总体来看查询方式主要分为两种:一种是非结构化查询;另一种是结构化查询。非结构化查询如 Gnutella、Freenet 等,采用的是泛洪方法,搜索的范围是用 TTL 参数来控制的,很显然搜索在超出一定范围后不能进一步扩展,而且搜索只限于以发起者为中心的某个特定半径范围以内。因此搜索并不能保证可靠的定位服务,即使在泛洪算法上进行性能改进,如采用随机泛洪算法等,查找的确定性仍是个问题。结构化路由如 Tapestry、Pastry、Chord、CAN 等,它们都是基于 DHT (Distributed Hash Table)技术,查找时只需要将关键词进行哈希计算,得到的结果就是放置文件或索引的结点 ID。因此该方式的查找具有确定性,而且具有可扩展性能好、容错性能高等优点。目前主流的基于 DHT 的 P2P 系统算法如下:

1) Tapestry 算法

Tapestry 是 B. Zhao、Kubiatowicz 和 Joseph[5,22~24]等提出的一种新型 Overlay 定位和路由算法,是目前主要非集中式定位和路由的研究项目之一。Tapestry 算法是对 PRR97 算法的改进和扩充,增加如下新特点:利用软状态信息(Soft State Information),提供自管理功能、强壮性、可扩展性、动态自适应性,在出现结点、网络失败和高负载时,实现平滑降低系统性能,不需要任意结点来维护全局信息,能够避免根结点的脆弱性和缺乏适应性。

Tapestry 是动态广域环境中对象命名、查找和消息路由系统的理想解决方案,通过点对点、无须集中服务的计算模型就可以发送消息,查找距离最近的对象拷贝。Tapestry 的主要实现方式是通过随机分布结点 NodeID 和对象 ObjectID 来取得负载平均分配,另外通过利用路由位置信息可以部分提高系统性能。

Tapestry 存在的性能缺陷在于:消息经常需要经过路由多个无用的底层 Internet 的自治系统(Autonomous Systems,AS)和管理域以后才能达到目的结点,一个 Overlay 路由往往会导致跨越多个 AS 系统,消耗网络带宽,使网络延时大幅上升,其根本原因就在于

Overlay 网络拓扑和路由是独立于底层 Internet 网络拓扑和路由的。

2）Pastry 算法

Pastry[25]是 Microsoft 公司和 Rice 大学共同发起的对等网络路由算法。在该算法中，每个结点有一个标识号 ID，并且维护一个 $\log_{2^b} n$ 行、2^b-1 列的路由表，一个邻居集和一个叶子集。

Pastry 的查找路由算法是根据“关键字”将信息路由到结点。查找时，首先在叶子集中查找是否存在目的结点，如果关键字落入叶子集中的 ID 范围内，信息就被转发到具有在数值上最近 ID 的结点，否则，从路由表中选择一个比现有结点有更多共同前缀的结点号，把查询转交给这个结点以便下一步查询。如果仍然不可能，则在叶子集中寻找与其 ID 具有相同前缀长度但数值是最接近关键字的结点。依此进行，直到找到目的结点。这种算法通常需要 $\log_{2^b} n$ 跳才能完成查找操作。

Pastry 与 Tapestry 有许多类似点：使用匹配前缀或者后缀地址的路由方法、插入和删除算法和类似的存储开销代价。Pastry 与 Tapestry 的区别在于：第一，Pastry 中对象复制无须由所有者来控制，在对象的发布过程中，对象经过复制后其复制对象放到与对象编号最接近的多个结点上；第二，Tapestry 在服务器和根结点之间路由转发过程中放置对象指针信息，而在 Pastry 中，客户端通过对象编号 ObjectID 直接路由到保存对象复制点附近，这样在实际网络中多个复制对象存在于实际物理网络的不同结点上，在多个结点增加存储开销，带来安全性、保密性和一致性等问题；第三，与 Tapestry 的 Surrogate Routing 算法相比，Pastry 的逻辑路由跳数界限具有较弱的理论分析性，Tapestry 在确定路由距离时具有可证明的理论分析性、完备定义性和概率界限，并保证能够找到已存在的可到达对象；第四，Pastry 的定位方法在即使对象位于客户端附近时，也需要连续路由，不能充分利用本地性，对本地对象定位可能带来较高的 RDP(相对平均延迟开销)值。

3）Chord 算法

Chord 算法[26]是由 MIT 提出的一种分布式查找协议，提供一种基于 DHT 的有效分布式查找服务。每个结点对应一个 m 位长度的名字空间，并保存该结点后面 $2^1, 2^2, \cdots, 2^m$ 个结点的指针信息，结点 n 路由表的第 i 个指针指向当前结点 n 后面第 2^{i-1} 位置的第一个结点，每个 key 保存在结点编号不小于该 key 编号的第一个结点上，每个结点仅需要 $O(\log n)$ (本书将 $\log_2 n$ 简记为 $\log n$)大小的存储空间来保存路由层的拓扑信息 Finger Table。当查找信息 key 时，首先计算 Hash(key)，然后在本结点的 Finger Table 中查找等于或小于但最接近 Hash(key)的结点，并把查询请求转发到该结点。然后递归执行以上步骤，直到最终查找成功。该算法的时间复杂度为 $O(\log n)$。Chord 算法提供与 Tapestry 算法类似的存储和逻辑路由跳数(Hop)界限。注意二者主要的区别是 Chord 算法中 Overlay 网络距离和底层物理网络距离无关，因此有可能在逻辑路由距离较近时，物理路由距离却较远。

4）CAN 算法

CAN(Content Addressable Network)[27]是由 Berkeley 提出的一种非集中式查找算法。CAN 模型将结点映射到 n 维的坐标空间，根据服务器的密度和负载情况，该坐标空间分成 n 维坐标空间区域，每个物理机器对应一个区域，并维护由哈希函数映射到该区域的数据和邻居信息。查找时，根据目标点坐标，从本结点开始沿着它的邻居点进行查询，直到找到目的结点。在 d 维空间中，每个结点维护 $2d$ 个邻居位置信息，即空间复杂度为 $2d$ 的平

均查找长度为$(d/4)(n^{1/d})$。

CAN 与 Tapestry 有多处差异，首先，Tapestry 的树形 Overlay 结构即使从网络中不同结点位置出发到达同一个目的结点能够迅速收敛，使用结点内部的位置信息和启发式功能使 Tapestry 的查询位置信息更加迅速，同时查询模式的适应性也可以通过上层应用来完成，而 CAN 假定对象不可改变，在对象改变后必须重新插入到 CAN 中；其次，Tapestry 网络拓扑使用本地网络延迟作为距离度量，近似接近实际物理网络，CAN 与 Chord 一样，在构造网络并没有完全考虑实际网络的距离，结果 CAN 的逻辑路由会导致高昂的代价，甚至逻辑邻居之间的一次路由转发会导致物理 IP 网络的多次路由。CAN 的主要优点在于结点加入算法的简洁性，能够更好地适应动态变化的网络环境(如传感器网络)。

综上所述，各自算法的性能如表 2-1 所示。

表 2-1　各种 P2P 路由算法的综合性能比较

P2P Systems	Napster	Gnutella	Pastry	Tapestry	Chord	CAN
Overlay 层构造	无	无	超立方结构	超立方结构	环	网格
空间复杂度	$O(n)$	依条件定	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(d)$
时间复杂度	$O(1)$	$O(1)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(n^{1/d})$
路由表项	—	—	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(d)$

5）小世界模型

为了解决 P2P 网络中的对等结点快速定位的问题，计算机科学家们借用并发展了社会科学家们针对于社会中的小世界现象提出的小世界模型。

1976 年，哈佛大学教授 Milgram[16] 做了这样一个实验，在内布拉斯加州的奥马哈随机选出 160 名当地居民，让他们每人设法将一封信传递到麻省的波士顿市的一个零售店主手中。这些人将信交给他认为最接近目标的一位朋友手中，接下来由这位朋友以同样的方式传递到下一位朋友手中，直到最后信送达目标人手中。最后有 42 封信到达了目标人手中，而且这些信的平均中间传递人数为 5.5 人。相比较 2 亿美国人口，这个数字显得非常低。这就是非常有名的“小世界”现象。

最早对小世界现象进行建模的是 Pool 和 Kochen[17]，该模型中每个结点均匀地从所有其他结点中随机地选择少量的结点然后连边。但是这个模型并不能很好地解释社会中的社区问题。也即尽管我们两个人相互认识，但你认识的绝大部分人我却都不认识，这点不符合社会常识。随后，Watts 和 Strogatz[18] 于 1998 年在《Nature》上提出了 Watts & Strogatz Model 模型，该模型认为相识边分为两种，每个结点拥有大量的近邻边(Local 边)，少量的远亲边(Long-Range 边)。远亲从远结点中均匀随机选择。正是这个模型的提出，使得 P2P 的应用研究有了理论指导。Freenet 主要就是依据该模型建立的。另外，在 2000 年，Jon Kleinberg 泛化了 Watts & Strogatz Model 模型，其在《Nature》上指出：该模型提出选择远亲时，概率应与离该结点的曼哈顿距离 d 有关，两点以概率 d^{-r} 连边。并给出了即使仅拥有距离为 1 的近邻和一个远亲，其算法的链长也不会超过 $O(\log^2 n)$。正是因为这篇文章作为坚实的理论依据，在 2001 年，一系列的非结构化、结构化算法被提了出来。

3. 新型 P2P 系统

CAN、Chord、Pastry、Tapestry 等系统都是确定性拓扑。最近为了进一步提高路由效

率，学者们又提出了一些新型 Overlay 层拓扑结构。

1）SkipNet

SkipNet 是 2003 年提出的新型的结构化 P2P 的拓扑架构。SkipNet[28] 使用了跳表(SkipList)的数据结构。SkipNet 架构采用了双标识的结构，为每个结点指定了 NumericID 和 NameID 两个标识以便更好地控制路由和拓扑结构。其中 NumericID 标识决定结点的分组情况以及结点与组之间的隶属关系；而 NameID 标识用来决定结点在每个路由环上的位置或坐标。所有结点以它所在的定位区域名被排序，然后以跳表方式形成统一的结构化拓扑。

SkipNet 这种标识自己分工的做法有效地提高了路由效率，减小了网络延迟，是新型 P2P 拓扑结构的代表。

2）Koorde

Koorde[63] 将数据分布于 de Bruijn 图中。每个结点有一个 t 位标识 $b_t b_{t-1} \cdots b_1$，并连接到标识为 $b_{t-1} b_{t-2} \cdots b_1 0$ 和标识为 $b_{t-1} b_{t-2} \cdots b_1 1$ 的两结点。它提供了“常量度路由”，t 增大有利于改善容错。

2.2　对等网络应用领域

P2P 计算引导网络计算模式从集中式向分布式偏移，也就是说网络应用的核心从中央服务器向网络边缘的终端设备扩散。这使人们在 Internet 上的共享行为被提到了一个更高的层次，使人们以更主动的方式参与到网络中去，每个人都能为网络的资源和功能扩展做出自己的贡献。

从目前的情况来看，P2P 计算主要应用在文件共享、普及计算、协同工作、搜索引擎以及广域网络存储等方面。

2.2.1　文件共享

文件共享是 P2P 最为流行的应用也是最为典型的应用之一，还是 P2P 技术革命的闪光点。Napster[29] 是提供给用户在 Internet 上共享 MP3 音乐文件的 P2P 应用，与传统的音乐共享技术不同的是 Napster 把音乐文件存储在客户结点上而不是存储在服务器结点上。中心服务器上存储的仅是文件的索引信息，用户之间可以直接共享、传输音乐文件而不需要通过中心索引服务器。采用这种方式来共享信息资源可以更加充分地利用网络中的带宽资源，从而提高了系统数据通信的效率。Napster[29]、Gnutella[21] 和 Freenet[12] 等文件共享系统被认为是 P2P 领域非常成功的范例。

2.2.2　普及计算

普及计算技术研究的是如何充分利用网络中各种各样的计算资源来共同完成大规模的计算任务。此类应用属于计算型资源共享，GRID[30] 和 SETI@Home 是该类型应用的典型代表。SETI@Home 是 Berkeley 大学启动的普及计算的研究项目。该项目是利用该大学的空间科学实验室开发的屏幕保护程序来使用空闲时的计算机，该屏幕保护程序在运行时分析在外星系文明研究项目中所获得的无线电信号，程序运行结点从中心服务器结点下载

数据后进行计算，然后再将计算结果上载到该实验室的中心服务器上，因为不是完全的 P2P 计算模式，所以结点之间不能直接利用彼此计算的数据。

2.2.3 协同工作

协同工作是指多个用户之间利用网络中的协同计算平台互相协同完成计算任务，共享信息资源等。Groove[31]是基于 Internet 的 P2P 协同应用软件的典型代表，其用户可以直接进行实时的协同工作。

2.2.4 广域网络存储

传统网络存储采用集群方式需要高容量高性能服务器，这需要一笔很大投资，而且缺乏本地特性，可扩展性也受到很大限制。P2P 技术提供了经济的、具有本地特性、灵活方便的可扩展功能，因而受到极大关注。特别是为了保障网络中存储的资源能够确定性查找，基于 DHT 的 P2P 系统方案是一个重要的选择。目前大部分 P2P 网络存储是基于 DHT 技术。下面介绍典型的网络存储应用。

OceanStore[32]提供了全球范围内的一致性数据存储，采用 Tapestry 作为底层的路由机制。它可以自动从服务器和网络的失效中恢复，并可以混合使用新的资源和适应不同的使用范式。PAST[33]是 Rice 大学和微软研究院的联合研究项目，它采用大规模协同的分布文件存储，提供可扩展性、可用性、安全性和协同资源共享，采用 Pastry 作为它的路由机制。它的存储特性是：文件所有者可以发布也可以收回文件的存储，提供的是一致性存储。CFS[34]是 MIT 的研究项目，它提供了一致性的分布协同文件存储，采用 Chord 作为它的路由机制。从文件系统的可读写角度看，PAST 系统仅支持只读，而 CFS 虽然基本上也是只读文件系统，但也存在粗粒度的修改，即文件所有者可以修改，而其余只能读。因此，可读写的文件存储系统是进一步的研究对象。IVY[35]是 MIT 最近的项目成果，它是一个多用户可读写 P2P 文件系统，基于 Chord 通信子层，适合广域的小型协作组。IVY 使得这些组避开了集中式文件服务器所固有的可靠性和信任问题。一个 IVY 文件系统只由一组日志文件组成，每一个用户对应一个日志文件。每一个参与者通过协商所有日志文件查询数据，但是修改则只是修改自己所有的日志文件。这种安排使得 IVY 可以不通过锁机制就可维护元数据一致性。当底层网络是满连接时，IVY 提供 NFS-Like 传统语义，如 Close-to-Open 一致性；当网络分区时，DHASH 层的复制使得修改可在分区内进行，并借助每一个日志文件所具有的版本标识实现分区合并时的一致性。Mnemosyne[36]是基于 Tapestry 的安全存储系统，仅合法的用户能够存取存储文件，而攻击者不能够得到它们的内容。文件通过编码技术加密存取，保证了容错性，编码技术如 Erasure Code[37]。

2.2.5 网页发布和缓存

传统网页发布和缓存是由高性能服务器提供的，存在 Flash Crowd 问题[38]。P2P 技术可以通过各对等结点的数据缓存而减轻请求热点，并且使得网页可以由全球结点共同提供，减轻了对服务器的依赖。DHT 技术特性使得定位自主，不必依赖协调器的定位服务（如 YouServ[39]），使其有更大的可扩展性和灵活性。

Squirrel[40]是一个基于 Pastry 的分散式的 P2P 网页缓存。系统聚集所有本地缓存形

成一个全球可扩展的网页缓存,而且并不需要附加的维护开销或者硬件投资。实验结果表明开发这样的一个系统不仅是可行的,而且比特定的网页服务器有更多的益处。

Coral[41]是纽约大学基于 DHT 的网页内容分发系统。每一结点均可复制并存储网页内容。所有结点的缓存协同工作,任何时候都尽可能将所需要内容从邻近结点得到,从而使得提供网页的最初服务器上负载最小化并尽可能减少客户端的延迟。

2.2.6　组通信

由于 DHT-P2P 系统中采用连通图拓扑来组织结点,因而适用于组通信。

Pastry 已经被用作 Scribe[42]的基础。Scribe 是一个事件通知体系,用于基于主题的出版/订阅系统。订阅者注册其感兴趣的主题并接收与该主题相关的事件。一个组播树用于维护每个相关于一个会合点(Rendezvous,比如叠加网上一个最接近主题标识的结点)的主题。每个树由通过加入从订阅者到特定主题的路由所组成。

DHT-P2P 系统也在应用层广播[43]以及应用层组播[44,45]得到应用,特别是应用层组播。应用层组播顾名思义就是在应用层实现组播功能而不需要网络层的支持。这样就可以避免出现由于网络层迟迟不能部署对组播的支持而使组播应用难以进行的情况。应用层组播需要在参加的应用结点之间实现一个可扩展的、支持容错能力的叠加网络,而大规模哈希表查找机制正好为应用层组播的实现提供了良好的基础平台。当前有两种途径用于组播:智能泛洪(Intelligent Flooding)和组播树(Multicast Tree)。前者用于基于 CAN 的 M-CAN[46],后者用于 Scribe[44]和 SplitStream[45],后者都是基于 Pastry。两种途径在相同的工作负载下的比较,显示组播树比泛洪途径性能优越[47]。

2.2.7　名字服务

DHT 机制通过哈希得到的键值对的方式及处理使得它非常适用于名字服务系统,使名字服务系统具有高效、准确、高可扩展等特性。

简单分布式安全体系(Simple Distributed Security Infrastructure,SDSI)是一个已提出的公钥体系,其中名字定义在本地的名空间(Namespace)并且长文件名能够链接多个名空间来代理使用认证的信任。ConChord[48]是一个基于 Chord 的分布式 SDSI 认证目录服务,用于多名空间的名字解释和成员检验,检查一个认证对于一个特定的钥(Key)是否可用。DHash[49]把 Chord 作为提供 DNS 服务的一个可选服务结构,DNS 中的"主机名、地址"对通过 DHT 技术能够以分布式方式存储,从而有利于减轻 DNS 树结构查找的负载不均衡、根结点瓶颈等问题。

2.2.8　信息检索

搜索引擎是目前人们在网络中信息检索的主要工具。目前的搜索引擎(如 Google、Yahoo 等)都是集中式的搜索引擎。这种信息搜索方式是一种被动的搜索方式,用户不能够主动将自己的信息发布到搜索引擎上,也不能够保持搜索引擎所采集数据的实时性。P2P 技术能够提供一种主动的、实时性、高扩展性的信息检索,开辟了信息检索的新方向。DHT 技术相比较于其他非结构化技术使得信息能够以一种确定的高效方式定位,因而引起特别的关注。

参考文献[50]较为全面地讨论了构造 P2P 搜索引擎所要面对的困难,并做出了可行性分析,特别是对于基于 DHT 技术的全文本网页搜索,提出了合理化的技术建议和一些创新的优化技术。PeerSearch[51]是第一个具有分散化、确定性和非泛洪的 P2P 信息搜索系统,可基于内容和语义进行搜索。PIER[52]提供了在 Internet 范围的 P2P 信息检索,能够提供基于关系查询的语义支持。

2.3 分布式哈希表与 P2P 系统

2.3.1 分布式哈希表简史和技术原理

分布式哈希表(Distributed Hash Table,DHT)作为可扩展的分布式数据结构(Scalable Distributed Data Structure,SDDS),在 20 世纪 90 年代得到广泛研究,SDDS 是在 Litwin 等的经典论文[53]中提出的。但是,这些分布式哈希表有中心部件,设计面向的是小规模的集群。面向大规模集群的分布式哈希表由 Gribble 等[54]采用 Java 实现,具有高度可扩展性、容错性和可用性。

分布式哈希表技术应用到对等网络是在 2001 年。一系列的基于分布式哈希表的 P2P 系统[55~58]被提出。这些系统采用分布式哈希表技术能够支持上百万的机器动态参与,具有高度的可扩展性和容错性。目前,基于分布式哈希表技术构造的 P2P 系统已经成为 P2P 研究的一项重要内容。

下面简单介绍一下 P2P 系统中分布式哈希表技术原理。

哈希表是计算机科学里常见的数据结构,它能够根据索引的关键码值快速查找记录。它通常提供两种基本功能:put 和 get 操作,也就是 put(key,value),value=get(key),且平均查找时间为常量度 $O(1)$。分布式哈希表是哈希表的分布式构造,将哈希表由单个结点扩展到 Internet 上。由于分布式哈希表是布置在整个 Internet 上,put 和 get 操作需要借助于分布路由而实现。哈希表与分布式哈希表的对照如表 2-2 所示。

表 2-2 哈希表与分布式哈希表对照

哈 希 表	分布式哈希表
Key=hash(data) Put(key,value) Get(key)－>value	Key=hash(data) Lookup(key)－>node_IP Route(node_IP,put,key,value) Route(node_IP,get,key)－>value

在 P2P 系统中是怎样采用分布式哈希表实现资源定位呢?

具体方法是首先将网络中的每一个结点分配虚拟地址标识 NodeID,同时用一个关键字 key 来表示其可提供的共享内容。取一个哈希函数,这个函数可以将 key 置换成一个哈希值 H(key)。网络中结点相邻的定义是哈希值相邻。发布信息的时候就把(key,NodeID)二元组发布具有和 H(key)相近地址的结点上去,其中 NodeID 指出了文档的存储位置。资源定位的时候,就可以根据 H(key)相近的结点,快速获取二元组(key,NodeID),从而获得文档的存储位置。不同的 DHT 算法决定了 P2P 网络的逻辑拓扑,比如 CAN 是一个超环,而

Chord 是一个环,Tapestry 是树状。

分布式哈希表提供给 P2P 系统设计以非常简洁高效的接口,然而,在实际应用中,分布式哈希表的设计要考虑如下三个关键问题:

(1) 唯一性:哈希函数必须避开碰撞或者碰撞概率非常小以至忽略不计。

(2) 动态性:哈希函数必须能够支持结点动态加入和离开的一致性。

(3) 合理尺寸:哈希表必须能够支持可扩展性,不能在单个结点上占用太大的空间。

对于第一个问题,可以通过定义一个固定的足够大的哈希空间,使所有哈希值落在此空间而不依赖结点的数目,从而在静态或者动态下能够避开碰撞。对于第二个问题通常采用的是一致性哈希方案。一致性哈希方案最初是由 MIT 的 Karger 等[59]引入以解决分布缓存中的热点问题,目前已经被扩展到很多领域,特别是 P2P 计算中。一致性哈希是哈希中的一种,但它具有如下特性使其特别适应动态分布的应用:可扩展性、负载均衡、平滑性。这些特性可以很好地适应动态性下的哈希正确性和一致性。对于第三个问题,解决的方法是将整个哈希表均匀分布到各结点上,每一结点负责它在哈希空间标识邻近的键值,因而需要结构化的拓扑组织以支持哈希表的分布及路由。故分布式哈希表构造是一种结构化的拓扑构造。

2.3.2　基于分布式哈希表的 P2P 系统/DHT-P2P 系统

DHT 技术对于 P2P 系统设计有革命性的影响。由于 DHT 支持具有有序的、高可扩展、可确定性查找的结构化拓扑,超越了之前的随机、ad hoc 的非结构拓扑,涌现了一大批的新型 P2P 系统,如 CAN[55]、Chord[4]、Tapestry[5]、Pastry[57]、Kademlia[60]、Symphony[61]、Viceroy[62]、Koorde[63]、P-Grid[64]等。这些系统都是采用 DHT 技术作为其设计基础。由于结点和数据都是通过分布式哈希表组织成一个叠加网络,从而这些系统被称为基于 DHT 的 P2P 系统,一些文献称其为 DHTs 或 DHT。为了明确和简化,本书称其为 DHT-P2P 系统,在不至于混淆情况下,有时本书也称其为 DHT 系统。

如 2.3.1 节所述,DHT-P2P 系统中每个结点负责一定范围的键值。叠加网中的结点既能够存储资源本身,也可能仅存储资源的地址指针,取决于算法需要。DHT-P2P 系统的路由是通过贪婪算法逐步逼近,具有在 Metric 空间最近距离的目标结点,这个 Metric 空间取决于系统的拓扑组织结构。

2.3.3　DHT-P2P 系统特性

DHT-P2P 系统是结构化系统,相比较于非结构化系统,它提供了如下几个很好的特性:

(1) 搜索不需要依赖于 Flooding 机制,因此造成较小的网络流量,大部分 DHT-P2P 系统中每个查询都只是需要 $O(\log n)$个消息和跳数。

(2) 每个查找请求都能以很高的概率解析,并且所需要的资源消耗是可预测的,而在非结构化系统里如果所请求的文档超越了查找所能覆盖的范围,则查询失败,而且即使查找成功,其资源消耗也不可预测。

(3) 搜索结果是确定性的。一方面,结构化拓扑数据只要存在叠加网络上就可以以高概率确定性查找到,而非结构化受 TTL 限制易查找失败;另一方面,查找结果也有确定性,

而在非结构化系统中,不同的结点提交同样的搜索请求时,很可能获得的结果也不同。

由于 DHT-P2P 系统这些的特性,使得它特别适合 Internet 分布应用,引起学术界的高度重视。

2.4 DHT-P2P 系统路由研究进展

路由算法是 P2P 系统的核心,算法的优劣直接关系到 P2P 系统的核心性能,如可扩展性、可靠性、可用性等。Sylvia Ratnasamy[65]等人在总结现有的 DHT-P2P 路由算法的基础上提出了结构化对等网络面临的“15 个问题”。这些问题体现在 5 个方面:状态效率折中、容错性、路由热点、物理网络匹配、异构性。事实上,它引导了对于 DHT 路由的研究方向,研究进展分述如下。

2.4.1 状态效率折中

Ratnasamy Sylvia 等[65]在 2002 年的 IPTPS 会议上介绍到,当前 DHT 路由有两种模式:

(1) 路由表大小为 $O(\log n)$、网络路径长为 $O(\log n)$,如 Tapestry、Pastry、Chord;

(2) 路由表大小为 $O(1)$、网络路径长为 $O(n_1/d)$,如 CAN。

这两种模式事实上反映了路由表空间与路由的路径长度的一种互为消长,Ratnasamy Sylvi 等称之为路由状态效率折中(Routing State-Efficiency Tradeoff)。其中状态指路由表所需要维护的邻居状态数目,效率指路由路径长度所代表的路由效率。一个重要的开放性问题是:能否实现结合以上两种模式优点的状态效率存在,即 $O(1)$的邻居数和 $O(\log n)$的路径长。这也被称为“常量度”路由问题。该问题引起了研究领域极大的兴趣,吸引了科研工作者对此深入地研究。最近已有一批常量度路由算法陆续提出,这些算法设计利用了不同拓扑几何的特性,如 de Bruijn[63]、Comb[66]支持常量度路由。但是,Jun Xu 等[67]分析指出,拥塞可能是这些路由算法的突出问题。此外,不同的研究方法也涌现,如传统构造默认路由表的分布一致性(Uniform),然而参考文献[68]采用不规则路由表,根据结点的能力来划分路由责任,对于状态效率的研究来说,是一种较为新颖的方法。利用结点的社会链关系[69,70]及随机路由算法[37]也是状态效率研究的新方法。由于状态效率对于 P2P 系统的性能具有本质性影响,时至今日,状态效率折中的研究仍然方兴未艾。

2.4.2 容错性

由于结点在 P2P 系统内自由加入和离开,容错性就显得特别重要。一方面,系统要在面对路由结点失效或者离开时,能够仍然保证路由可行性和正确性;另一方面,在大规模失败时,网络可能由于结点间完全失去连接,形成孤岛现象,这种孤岛现象也要处理好。对于前者的评估,Sylvia Ratnasamy 提出静态弹性(Static Resilience)概念。静态弹性是指当处于路由过程中的结点失效,但系统并不做路由修复时(如选择新的结点替换相应的路由表项),路由的可行性及效率(这个问题在参考文献[71]中得到了较好的回答)。参考文献[71]认为静态弹性与路由 Geometry 密切相关和环结构对于静态弹性有效,此外还区分路由表的邻居为规则和持续两类,其中持续邻居对于静态弹性更有效。相似的结论出现在参考文

献[72]中，不过，此时是将规则和持续这两类对应为长链和短链，并通过随机过程理论分析和实验得出短链对于静态弹性有更重要的影响。参考文献[73]建立了考虑有一半系统内结点失效时系统的重建过程的分析模型，参考文献[74]利用 SkipNet 的特性考虑分区问题，但目前如何有效处理大规模失败的问题仍然是一个开放性问题。

2.4.3 路由热点

路由热点是指当存有资源的结点受到太多请求时，此结点的出入路由流量太大，引起所谓“热点”问题，即路由拥塞。此时，结点由于负载过重而反应不过来。路由热点的解决方案一方面可以通过有效地复制和缓存策略[38,75]，以分流对于热点资源的请求，另一方面采用负载均衡技术。在 DHT 中，负载均衡可采用虚拟服务器(Virtual Server)的方法，根据结点的能力分配负载，如 Chord[4]。在参考文献[76]中，基于 Chord，进一步考虑了虚拟服务器根据动态负载的情况进行迁移(Transfer)的技术，其技术要点是重载结点将一些虚拟服务器迁移到轻载结点，并考虑了总体性能的均衡，而参考文献[77，78]则继续对于减少迁移的开销做了些改进和相应的理论分析。参考文献[79]采用的 the Power of Two Choices 技术在哈希发布文档时，就将文档索引项采用多个哈希函数进行了多个结点的存储，并在查询时随机选取一个存储结点进行定位路由，从而较好地均衡了负载。

2.4.4 物理网络匹配

物理网络匹配指的是叠加网络应尽可能与物理网络相匹配以减少结点通信开销和路由延迟。目前的研究主要是从如下两个互为补充方向进行：

1. 利用 Internet 的层次信息(如自治系统、IP 前缀)进行拓扑适应构造

这是一种直接与物理网络匹配的方法。

eCAN[80,81]借助 BGP 表来引导路由，使得结点通信本地化，减少通信开销和路由延迟。而参考文献[82]利用 IP 前缀使得路由充分利用 Internet 层次信息，并模拟路由层的最短路径算法以匹配物理网络。参考文献[83]提出一种 Sloppy 哈希技术以支持邻近路由。参考文献[84]则是根据结点的 IP 时延将叠加拓扑组织成多个层次，路由根据时延由短到长的层次进行，以尽可能减少整体通信时延。这些方式与物理网络的匹配性都是需要在设计时就考虑好，主要是基于叠加拓扑构造的方法。

2. 采用路由选择技术

这是一种间接与物理网络匹配的方法，DHT 路由选择技术有如下三种[85,86]：

(1) 邻近邻居选择(Proximity Neighbor Selection，PNS)：在路由表构造时，结点选择与该结点邻近(如时延较短、物理位置靠近)的结点作为邻居。这种优化策略能使路由跳转充分选择结点的物理邻近结点进行，从而使得路由间接与物理网络匹配。该技术的成功依赖于结点在依此构造路由表时能够不影响总体路由跳数的自由度。在基于前缀的协议中(如 Tapestry 和 Pastry)，路由表的上层允许更自由的选择，而下层的选择自由度呈指数下跌。因此，第一跳的平均延迟非常低，而随着每一跳指数增加，最后一跳的延迟决定着整个查找过程的延迟。该类方法有好的延迟伸展、负载平衡和本地路由收敛特性[85]，但是该类方法的限制不支持如 CAN 和 Chord 这类要求在路由表中明确指出标识空间的下一个结点标识的 DHT 算法。

(2) 邻近路由选择(Proximity Routing Selection,PRS):在路由时,选择与该结点邻近的邻居作为下一跳。这种优化路由以匹配物理网络的构造技术是基于如果每步都是最短时延,那么总体时延也应该得到优化的原理。与 PNS 相反,它是在路由过程中进行的动态选择。假设每一跳都有 k 个结点可供选择,则每一跳的平均延迟可以从原来的网络中任意两个结点延迟的平均值减少到网络中任意一个结点到其他任意 k 个结点延迟最小值的平均值,所获得的性能提高同 k 的大小成正比,增大 k 的值意味着每个结点路由表的增大,从而需要更多的资源消耗以维持链接。除此之外,一味考虑选择延迟最低的结点转发查询也可能导致路由逻辑跳的数目变大。

(3) 邻近标识选择(Proximity Identity Selection,PIR):在新结点加入时,结点标识的产生与结点所在的物理位置相关。它的原理是基于如果标识与物理布置相关,那么在以标识为基础的叠加网的路由就能够尽可能接近以 IP 为基础的物理网的路由。Landmark[55]技术广泛应用于 PIR。其方法是取 m 路标,然后每个加入的结点通过 ping 这 m 路标得到的值排列成 m 位,即为结点标识。由于此标识反映了与 m 路标的物理位置关系,因此,将标识与物理位置结合了起来。Brocade[87]是基于 PIR 的路由技术,eCAN[80]也采用了这种技术。参考文献[88]实现了 PIR 路由,它在每一跳获得了 IP 延迟的两倍或者更少的性能。然而,该方法有很多缺点:首先,它破坏了标识空间的均匀分布,在叠加网中造成了严重的负载不平衡问题;其次,由于映射算法的限制,该方法在一维空间中(Chord、Tapestry 和 Pastry)工作效果较差;再次,标识空间中相邻的结点由于物理位置也相邻,容易造成并发失败,而由于在 Chord 和 Pastry 这样的系统中结点在邻居中复制数据,因此也易造成安全性和鲁棒性隐患。

以上三种方法中,PRS 是最轻量级的方法,但是性能受到 k 的限制,升高 k 值同时导致叠加拓扑的维护耗费更大。参考文献[71]实验结果表明,PNS 方法的性能优于 PRS,但受限于一些明确下一跳的 DHT 算法(如 Chord)。PIS 具有较好的平均每跳延迟,但是该种方法导致严重的负载不平衡并需要高维标识空间才能有效。

2.4.5 异构性

当前结构化 P2P 系统清晰或不清晰地假定所有结点在资源方面(网络带宽、存储和 CPU)是一致分布的。消息在叠加网路由时并不考虑参与结点的能力差异。然而,测量研究表明 P2P 系统有极大的异构性[89]并且由于一部分结点非常有限的能力瓶颈可能引起路由算法失效。因此,将结点异构性考虑进去。利用异构性可以分配更多的能力给有高网络带宽、大存储容量和好的 CPU 处理能力的结点,这些结点称为超级结点。参考文献[90]建立了一个超级结点虚拟层,并将本地结点组织成一个以超级结点为中心的组,并采用两阶段的路由过程,第一阶段是将请求路由给超级结点层,第二阶段时再将请求路由给超级结点所在组的目标结点。这种方式可以避开低能力结点的瓶颈,提高通信效率,降低通信延迟。参考文献[91]通过异构结点形成层次化 DHT 路由以得到与参考文献[90]相似的效果。

2.5 DHT-P2P 系统拓扑研究进展

拓扑对于 P2P 系统性能有重要的影响,直接与路由及查询性能相关。P2P 拓扑应该能够适应动态的网络环境的变化并与物理网络有更紧密的结合,增加 P2P 系统的适用范围和

提高 P2P 系统的性能和效率。DHT-P2P 系统提供了结构化网络拓扑，目前的研究主要集中在以下三方面：

(1) 控制拓扑维护开销：增强 DHT 拓扑的网络动态适应性。

(2) 层次化拓扑：克服 DHT 的平坦化结构，加入现实中的层次特征。

(3) 混合拓扑：结合结构化与非结构化拓扑的两种拓扑的优点。

这三方面的研究抓住了 DHT 技术本身的特点，以更好地提高 DHT 拓扑的适应性和效率。

2.5.1 控制拓扑维护开销

DHT-P2P 系统由于将结点和文件紧密布置在一个结构化拓扑中，从而敏感于结构的动态变化。为了保障在动态网络环境下的正确结构和路由正常进行，维护拓扑的开销相对较大，特别是极动荡时可能超过系统的控制。

结点不断加入和离开的过程称为 Churn，它对于系统性能有重要的影响。Sean Rhea 等[92]研究了在 Churn 情况下性能的影响因子，并通过主动失败恢复、合理超时设置、邻近邻居选择等技术较好地控制了 Churn。Bamboo[93]基于这些技术较好处理了 Churn 的一个 DHT-P2P 系统。参考文献[94]比较了 Churn 情况下不同的 DHT(Tapesty、Chord、Kelips、Kademlia)的性能，得出在同样负载下，如果参数调整得足够好，它们都具有相似的性能。然而，参数的调整在不同的环境及协议下也是不同的，要具体情况具体分析。参考文献[95]建立了结点开销模型以用于观察系统的稳定性、热点及网络效率。参考文献[96]从路由表的构造上分析了开销减少的可能性。参考文献[97]分析了真实 trace 下的拓扑维护开销，并通过自反馈机制来调整性能，减少动态性对于系统性能的影响。参考文献[98]讨论了拓扑维护的现实性和意义。

2.5.2 层次化拓扑

DHT 设计面向一个平坦(Flat)的拓扑。它的优点是对于所有参与结点有全局一致的功能分布，并且没有单点故障。然而，层次性的缺乏使得特定系统的层次信息(如目录层次结构、层次缓存等)丢失，且管理较为困难。一些研究者提出了层次化拓扑的设计思想。Canon[99]是基于 DHT 的一种层次化拓扑设计，不仅保持原有 DHT 的状态路由效率，而且具有以下五点特性：①错误隔离；②用于多播的有效缓存和有效带宽；③匹配物理网络的拓扑；④层次化内容存储；⑤层次化存取控制。TerraDir[100]提供了层次名空间(如 DNS 或 UNIX 文件系统名)的目录查询。它组织结点为层次树结构，父结点维护它的子结点的信息。为了容错和减少查询延迟，缓存和复制在系统中大量使用。参考文献[101]提出了一个通用的框架用于 P2P 叠加网络的层次组织。聚簇是实现层次化的常用方式。参考文献[82]根据网络 IP 前缀进行分层次聚集的层次化 DHT 设计。参考文献[102]中的观点是当前网络物理拓扑结构是层次化，因而设计层次化 DHT 以更好地匹配物理网络和利用本地的局部特性。

2.5.3 混合拓扑

混合拓扑是新兴的一个研究点。结构化和非结构化各有其优缺点，混合网络拓扑综合

了两者性能优点，减轻了其性能缺点，是一种有益的探索。Yapper[103]有一个与 Gnuetlla 相似的非结构的网络拓扑，但是引进哈希来存储数据键。每结点 x 被分配 b 种颜色：Color(x)＝Hash(IP(x)) mod b。每个结点维护它的直接邻居（距离结点为一常量值的路由跳数 c 的结点）信息。资源存储在颜色与资源键哈希值（为一颜色）匹配的结点上，因而查找可限定在同种颜色的结点上，从而查询性能得到较大改善。Kelips[104]在 DHT 基础上引入集中式组管理。其中 n 结点通过哈希聚簇 $O(\mathrm{sqrt}(n))$ 仿射组。每个组内有一超级结点维护组内所有信息并由别的组进行通信。每个结点的空间占用是 $O(\mathrm{sqrt}(n))$，路由需要 $O(1)$ 时间。网络更新消息采用 Gossip 协议[105]传播。不过，在超大规模的 P2P 系统里，这种组维护开销可能过大。LOO 等人[106]在 IPTPS04 会议提出一种新的设计思想：结构化适合定位查找稀罕数据项(Rare Item)，而非结构化适合定位查找流行数据项(Popular Item)，因而设计混合拓扑模型。网络中一部分为结构化 DHT 构造采用 DHT 方式定位索引稀罕数据项，另一部分为非结构化 Gnutella 构造采用泛洪方式定位通用数据项。LOO 等的实验表明此种混合方式使得查询效率和可扩展性都得到了较好的提高。

2.6 DHT-P2P 系统查询研究进展

查询是 P2P 系统中的一个关键问题，也是 P2P 系统最广泛的应用。参考文献[107]对于 P2P 查询的问题做了一个总结，不过目标是非结构化系统。对于结构化系统，它具有与非结构化很不相同的问题。最根本的原因是由于 DHT 采用哈希技术仅提供精确查询匹配，使得 DHT 查询受到极大的约束。DHT 查询的前景展望见参考文献[108]。

DHT 查询的研究主要从以下三个方面展开，它们的目的是增强 P2P 查询能力，从而扩展 P2P 系统的应用范围。

(1) 多关键字查询。

(2) 模糊关键字查询。

(3) 复杂查询。

2.6.1 多关键字查询

当前基于 DHT 的 P2P 系统为了能够支持多关键字的数据检索，一般首先对文档做索引，一般采用逆序索引(Inverted Index)方式，然后针对每个关键字，将“关键字、文档标识”插入到 DHT 叠加网中。

当进行多关键字组合查询时，针对每个关键字，根据 DHT 算法映射到存储该关键字的相应结点，在该结点上查询包含该关键字的文档列表，然后在结点之间顺序传送文档列表并计算交集。显然这种检索机制需要在网络上传输大量的中间文档列表数据，产生大量的网络带宽消耗，当前有许多研究集中在如何减少带宽消耗以及提高检索效率上。在减少带宽消耗方面，参考文献[109]采用了 Bloom Filters[110]机制，如图 2-2 所示。例如，进行 $A \cap B$ 组合查询，设关键字 A 映射到结点 A，关键字 B 映射到结点 B。查询请求首先发往结点 A 检索包含关键字 A 的文档集合。在结点 A 上检索到包含关键字 A 的文档集合，然后计算该集合的 Bloom Filter 并将结果 $F(A)$ 发往结点 B，结点 B 检索本地得到包含关键字 B 的文档列表，并利用 $F(A)$ 计算出满足 Bloom Filter 测试的 B 文档集，记为 $F(A) \cap B$，该结果

集再传递回给结点 A,并计算和 A 文档集的交集,将结果发送回客户端。

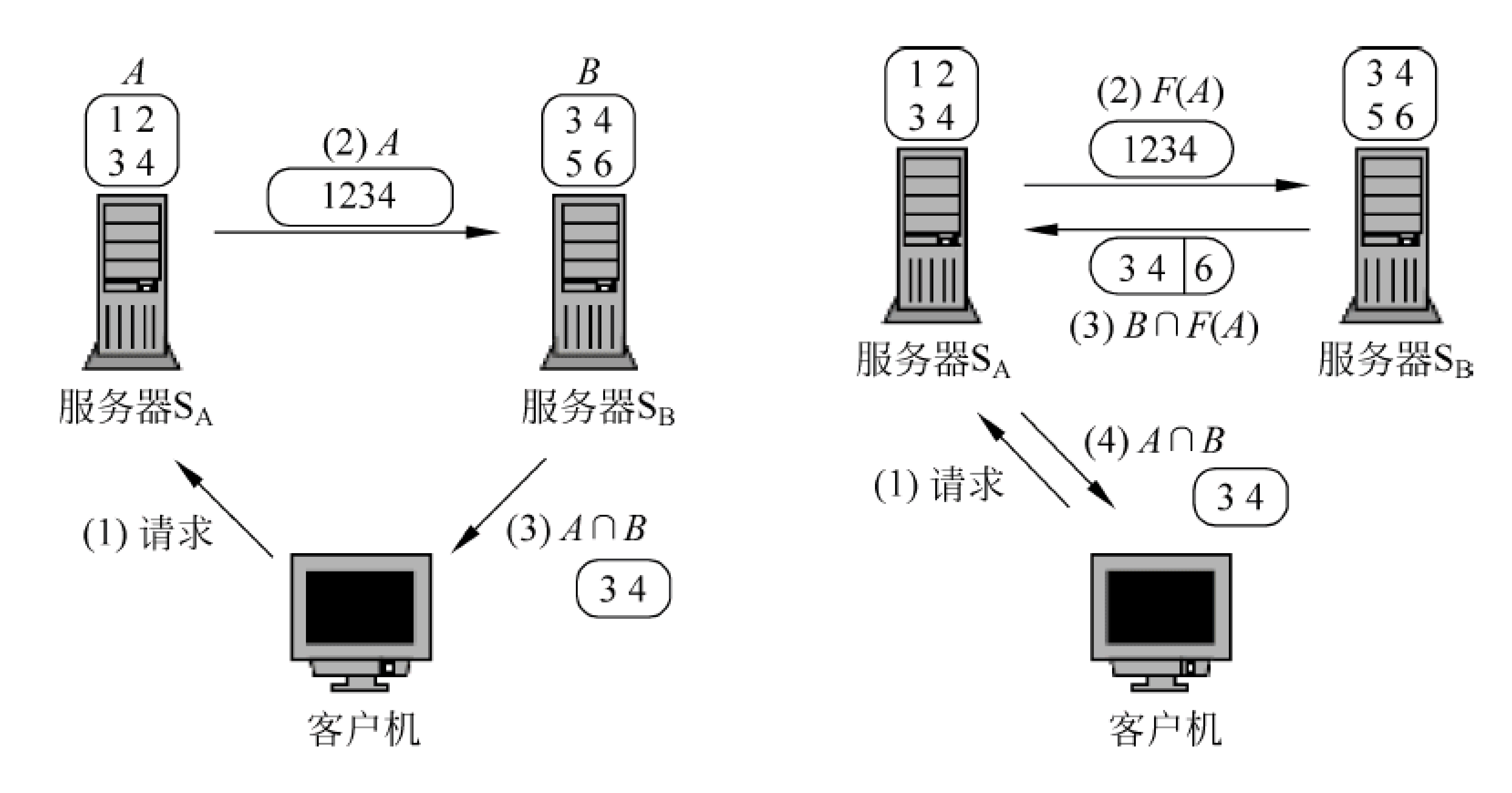


图 2-2　使用 Bloom Filter 计算组合查询 $A \cap B$[109]

如果用户数据检索的要求允许一定的误差,在上述过程中,结点 B 可以直接将 $F(A) \cap B$ 发送给查询客户端,这可能会包含一些"正向错误"(即多包含了一些不正确的结果),Bloom Filter 的位数越多,正向错误率越低,但消耗的带宽越高。

在多关键字查询中,其中的一个关键问题是如何减少结果传输对带宽的占用。如前面所介绍,Bloom Filter 是常用的技术,此外,还有其他一些可采用的技术。参考文献[111]利用了每次查询后的结果缓存(Result-Caching)技术来避免重复查询和减轻通信流量。例如针对三个索引 a_i、a_j、a_k,查询 $a_i \cap a_j \cap a_k$ 直接从索引 a_i、a_j、a_k 得到显然比利用先前 $a_i \cap a_j$ 的查询结果缓存再与 a_k 的交集计算得到所花费的代价大得多。参考文献[112]提出了 View Tree 来存储和查询以前的结果缓存。此 View Tree 实质是一个 Trie 结构,通过此能够很好地减少多关键字的查询开销。

以下这些技术并不直接相关于多关键字查询,但是其思想可以为多关键字查询所借鉴。

参考文献[113]提出部分查询技术以及相应的内容模式。其思想是考虑用户查询时并不需要所有拷贝,而是需要一部分就够了,比如查询一个流行歌曲只需要联系两三个网站就够了,而并不需要可能的上百个网站。因而,相对于传统查询返回所有结果集,部分查询能够有助开销减少和查询效率的提高。特别是它不敏感于流行 key,能够有效减轻热点问题。参考文献[114]提出了轻索引技术,其思想是考虑建立索引的目的是为了有效进行查找,但并不是所有的内容都要索引,因而仅索引那些根据反馈计算得出值得索引的那些内容,从而降低系统的索引维护开销。部分查询技术和轻索引技术均能够有助于多关键字查询中的开销的减少,并有望提高查询效率。

2.6.2　模糊关键字查询

为使基于 DHT 的 P2P 系统能支持模糊查询,Harren 等[108]提出了一种采用 n-grams 的方法解决模糊查询问题,如 thoven 可分解为 tho、hov、ove、ven 等 3-grams(长为 3 个字母的片段)。通过针对不同的值 n,可以分别建立索引。这种方法可以支持模糊查询,但是它占用存储容量、网络带宽,以及处理开销都太大,扩展性很差。目前仍然没有很好的方案来解决这个问题,因此还有不少挑战性的工作需要去做。

2.6.3 复杂查询

应用查询应不仅支持关键字查询，还应支持关系数据库的 SQL 查询（如选择(Selection)、投影(Projection)、联合(Join)、聚集(Group-by/Aggregate)等）及语义支持的 IR 模型查询等复杂查询。

参考文献[108]首次提出了 DHT 上 SQL 查询的设计思想，但仅实现了联合(Join)。PIER[52]实现了一个 SQL 查询子集，但是在使用中经常会出现查询结果分布不均匀、大量的"热区"的现象。作为 SQL 初步查询，范围查询是目前 SQL 查询的热点研究。参考文献[115]基于二维 CAN 提出了一种范围查找的方法，它利用了 CAN 的空间位置信息来存储查询结果集合，并由此支持后续的子范围查询。Gupta 等[116]使用位置敏感哈希(Locality-Sensitive Hashing，LSH)[117]来支持近似(Approximate)范围查询。位置敏感哈希 LSH 是指哈希函数集合 H，$\forall h \in H$，有 $P_r[h(A)=h(B)]=\mathrm{sim}(A,B)$。其中 A 和 B 为两个集合，$\mathrm{sim}(A,B)$代表此两个集合的相似度。它们用 LSH 代替了 Chord 中的一致性哈希，从而使得相似的数据项能够在哈希空间也接近。查询一些数据项将产生相似于被请求项的结果，因而支持范围查询。但是由于采用了不同的哈希功能代替一致性哈希，系统的负载均衡受到了影响。

参考文献[118]提出元数据搜索层(Meta Data Search Layer)用于统一数据和文件的描述和定义。事实上，这种建立元数据信息描述的手段已经在 Web Service 和服务发现机制中得到较为广泛的研究和应用。元数据搜索具有比简单数据搜索更丰富的语义。SOMO[119]借助于元数据来管理网络资源。Arturo Crespo 等[120]引入了路由索引(Routing Indices)的概念，允许邻居结点传递查询请求到更可能回答此请求的结点。如果一个结点不能够回答查询，它传递此查询到基于本地路由索引的一个邻居结点集，而不是随机选择邻居结点或者泛洪此请求给所有邻居结点。借助于路由索引，查询性能得到较好的提升，但它也需要额外的存储空间开销。

参考文献

[1] Karl Aberer，Manfred Hauswirth. Peer-to-Peer Information Systems：Concepts Models，State-of-the-Art，Future Systems. Tutorial at The 18th International Conference on Data Engineering (ICDE)，San Jose，California，2002.
[2] Mike Miller. Discovering P2P. Sybex International，2001，11. ISBN-0782140181.
[3] Peer-to-Peer working Group Committees. http://peer-to-peerwg. org.
[4] Stoica I，Morris R，Kargea D，Kaashoek M F，Balakrishnan H. Chord：A scalable peer-to-peer lookup service for internet applications. ACM SIGCOMM，2001.
[5] Zhao B，Kubiatowicz K，Joseph A. Tapestry：An infrastructure for fault-resilient wide-area location routing. Technical Report UCB//CSD-01-1141，University of California at Berkeley，2001，4.
[6] Rowstron A，Druschel P. Pastry：Scalable，distributed object location routing for large-scale peer-to-peer systems. Lecture Notes in Computer Science，2001，2218：329-350.
[7] Ratnasamy S，Francis P，Hley M，Karp R，Shenker S. A Scalable Content-Addressable Network. Proc. ACM SIGCOMM 2001，2001，8.
[8] Stephanos，routsellis-Theotokis，Diomidis Spinellis. A Survey of Peer-to-Peer File Sharing

Technologies. White paper, Electornic Trading Research Unit (ELTRUN), Athens University for Economics Business, 2002. http://www.eltrun.aueb.gr/whitepapers.

[9] Oram A. Peer-To-Peer: Harnessing the Power of Disruptive Technologies. O'Reilly, first edition, 2001, 3. ISBN: 0-596-00110-X.

[10] Napster. http://en.wikipedia.org/wiki/Napster.

[11] Gnutella. http://gnutella.wego.com.

[12] Clarke I, Sberg O, Wiley B, Hong T W. Freenet: A distributed anonymous information storage retrieval system. In ICSI Workshop on Design Issues in Anonymity Unobservability, Berkeley, California, 2000.

[13] Markatos E P. Tracing a Large-Scale Peer to Peer System: An Hour in the Life of Gnutella. In The Second International Symposium on Cluster Computing the Grid, 2002. http://www.ccgrid.org/ccgrid2002.

[14] Petar Maymounkov, David Mazires. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In The 38 BIBLIOGRAPHY 1st International Workshop on Peer-to-Peer Systems (IPTPS'02), 2002.

[15] KaZaA. http://www.kazaa.com.

[16] Milgram S. The small world problem, Psychology Today, 1967, 1: 61.

[17] Pool I, Kochen M. Contacts influence. Social Networks, 1978, 1: 1-48.

[18] Watts D, Strogatz S H. Collective Dynamics of 'Small-World' Networks, Nature, 1998, 393, 440-442.

[19] IRIS Project. http://project-iris.net.

[20] Dingledine R, Freedman M J, Molnar D. The free haven project: Distributed anonymous storage service. In: Proc. of the Workshop on Design Issues in Anonymity Unobservability. 2000. 67-95.

[21] Gnutella. http://gnutella.wego.com.

[22] Zhuang S Q, Zhao B Y, Joseph A D, Katz R H, Kubiatowicz J D. Bayeux: An architecture for scalable fault-tolerant widearea data dissemination. In Proceedings of the 11th International Workshop on Network Operating System Support for Digital Audio Video, ACM, 2001, 6.

[23] Zhuang S Q, Zhao, B Y, Joseph A D, Katz, R H, Kubiatowicz J D. Bayeux: An architecture for scalable fault-tolerant widearea data dissemination. In Proceedings of the 11th International Workshop on Network Operating System Support for Digital Audio Video, ACM, 2001, 6.

[24] Weatherspoon H, Wells, C, Eaton P R, Zhao B Y, Kubiatowic J D. Silverback: A global-scale archival system. ACM SOSP, 2001.

[25] Druschel P, Rowstron A. Pastry: Scalable, distributed object location routing for large-scale peer-to-peer systems. ACM SIGCOMM, 2001.

[26] Stoica I, Morris R, Kargea D, Kaashofk M F, Balakrishnan H. Chord: A scalable peer-to-peer lookup service for internet applications. ACM SIGCOMM, 2001.

[27] Ratnasamy S, Francis P, Hley M, Karp R, Schenker S. A scalable content-addressable network. In Proc. of SIGCOMM, ACM, 2001, 8: 161-172.

[28] Harvey N J, Jones M B, Saroiu S, Theimer M, Wolman A. Skipnet: A scalable overlay network with practical locality properties. In Proc. of USITS, USENIX, Seattle, WA, 2003, 3: 113-126.

[29] Napster: http://www.napster.com.

[30] GRID. http://www.grid.com.

[31] Groove. http://www.groove.net.

[32] Kubiatowicz, J, Bindel D, Chen Y, Eaton P, Geels D, Gummadi R, Rhea S, Weatherspoon H, Weimer W, Wells C, Zhao B. OceanStore: An architecture for global-scale persistent storage. In Proceedings of ACM ASPLOS ACM, 2000, 11.

[33] Rowstron A, Druschel P. Storage management caching in PAST, a large-scale, persistent peer-to-peer storage utility. In Proc. of SOSP, ACM, 2001, 10: 188-201.
[34] Dabek F, Kaashoek M F, Karger D, Morris R, Stoica I. Wide-area cooperative storage with CFS. In Proc. of SOSP (Oct 2001), ACM, 2001, 10: 202-215.
[35] Muthitacharoen A, Morris R, Gil T M, Chen B. Ivy: A read/write peer-to-peer file system. In Proc. of OSDI (Dec 2002), ACM, 31-44.
[36] Roscoe T. Mnemosyne: Peer-to-peer steganographic storage. In Proc. of IPTPS. 2002, 3: 130-140.
[37] Hakim Weatherspoon, John D Kubiatowicz. Erasure Coding vs. Replication: A Quantitative Comparison, Electronic Proceedings for the 1st International Workshop on Peer-to-Peer Systems (IPTPS'02).
[38] Tyron Stading, Petros Maniatis, Mary Baker. Peer-to-Peer Caching Schemes to Address Flash Crowds Crowds, 1st International Peer To Peer Systems Workshop (IPTPS 2002).
[39] Roberto J Bayardo Jr., Rakesh Agrawal, Daniel Gruhl, Amit Somani. YouServ: A Web-Hosting Content Sharing Tool for the Masses. In Proceedings of the Eleventh International World Wide Web Conference (WWW), Honolulu, HI, USA, 2002, 5: 345-354.
[40] Sitaram Iyer, Antony Rowstron, Peter Druschel. SQUIRREL: A decentralized, peer-to-peer web cache. In Proceedings of the Twenty-First ACM Symposium on Principles of Distributed Computing (PODC), Monterey, CA, USA, 2002, 7: 213-222.
[41] Freedman M J, Freudenthal E, Mazires D. Democratizing content publication with coral. In Proc. of NSDI, 2004, 3.
[42] Rowstron A, Kermarrec A M, Druschel P, Castro M. SCRIBE: The design of a large-scale event notication infrastructure. In Proceedings of the Third International Workshop on Networked Group Communication (NGC), volume 2233 of Lecture Notes in Computer Science, UCL, London, UK, 2001, 11: 30-33.
[43] Sameh El-Ansary, Luc Onana Alima, Per Br, Seif Haridi. Efficient Broadcast in Structured P2P Networks. In Proceedings of the Second International Workshop on Peer-to-Peer Systems (IPTPS), Berkeley, CA, USA, 2003, 2.
[44] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Antony Rowstron. SCRIBE: A large-scale decentralized application-level multicast infrastructure. IEEE Journal on Selected Areas in Communications (JSAC) (Special issue on Network Support for Multicast Communications), 2002, 10, 20(8): 1489-1499.
[45] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Animesh Ni, Antony Rowstron, Atul Singh. SplitStream: High-bwidth content distribution in a cooperative environment. In Proceedings of the Second International Workshop on Peer-to-Peer Systems (IPTPS), Berkeley, CA, USA, 2003, 2.
[46] Ratnasamy S, Hley M, Karp R, Shenker S. Application level Multicast using Content-Addressable Networks. In Proceedings of the Third International Workshop on Networked Group Communication (NGC), volume 2233 of Lecture Notes in Computer Science, UCL, London, UK, 2001, 11: 14-29.
[47] Miguel Castro, Michael B Jones, Anne-Marie Kermarrec, Antony Rowstron, Marvin Theimer, Helen Wang, Alec Wolman. An Evaluation of Scalable Application-level Multicast Built Using Peer-to-peer Overlays. In Proceedings of the Twenty-Second Annual Joint Conference of the IEEE Computer Communications Societies (INFOCOM), San Francisco, CA, USA, 2003, 3.
[48] Sameer Ajmani, Dwaine E Clarke, Chuang-Hue Moh, Steven Richman. ConChord: Cooperative SDSI Certicate Storage Name Resolution. In Peter DruSCHEL, FRans Kaashoek, Antony Rowstron, editors, Proceedings of the First INTERNATIONAL Workshop on Peer-to-Peer Systems (IPTPS), volume 2429 of Lecture Notes IN COMPUTEr Science, Cambridge, MA, USA, 2002, 3: 141-154.

[49] Cox R, Muthitacharoen A, Morris R T. Serving DNS using a Peer-to-peer Lookup Service. In Peter Druschel, Frans Kaashoek, Antony Rowstron, editors, Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS), volume 2429 of Lecture Notes in Computer Science, Cambridge, MA, USA, 2002, 3: 155-165.

[50] Li J, Loo B T, Hellerstein J, Kaashoek F, Karger D, Morris R. On the Feasibility of Peer-to-Peer Web Indexing Search. In Proceedings of the Second International Workshop on Peer-to-Peer Systems (IPTPS), Berkeley, CA, USA, 2003, 2.

[51] Tang C, Xu Z, Mahalingam M. pSearch: Information Retrieval in Structured Overlays. In HotNets-I, October 2002. Exped version available as HP technical report HPL-2002-198. Peer Search: Efficient Information Retrieval in Peer-to-Peer Networks.

[52] Huebsch R, J, Hellerstein M, Lanham N, Loo B T, Shenker S. Stoica I. Querying the Internet with PIER. In Proc. 19th VLDB, 2003, 9.

[53] Litwin W, Neimat M A, Schneider D A. LH *--A scalable, distributed data structure. ACM Transactions on Database Systems, 1996, 21(4): 480-525.

[54] Gribble S D, Brewer E A, Hellerstein J M, Culler D. Scalable, distributed data structures for Internet service construction. Proc. 4th Symposium on Operating System Design Implementation (OSDI 2000), 2000: 319-332.

[55] Ratnasamy S, Francis P, Hley M, Karp R, Schenker S. A scalable content-addressable network. In Proc. of SIGCOMM, ACM, 2001, 8: 161-172.

[56] Zhao B Y, Kubiatowicz J D, Joseph A D. Tapestry: An infrastructure for fault-tolerant wide-area location routing. Tech. Rep. CSD-01-1141, U. C. Berkeley, 2001, 4.

[57] Rowstron A, Druschel P. Pastry: Scalable, distributed object location routing for large-scale peer-to-peer systems. In Proc. of Middleware, ACM, 2001, 11: 329-350.

[58] Stoica I, Morris R, Karger D, Kaashoek M F, Balakrishnan H. Chord: A scalable peer-to-peer lookup service for internet applications. ACM SIGCOMM, 2001.

[59] Karger David, Lehman Eric, Leighton Tom, Levine Matthew, Lewin Daniel, Panigrahy Rina. Consistent Hashing Rom Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. In Proceedings of the Twenty-Ninth ACM Symposium on Theory of Computing (STOC), El Paso, TX, USA, 1997, 5: 654-663.

[60] Maymounkov P, Mazieres D. Kademlia: A peer-to-peer information system based on the XOR metric. In Proc. of IPTPS, 2002, 3: 53-65.

[61] Manku G S, Bawa M, Raghavan P. Symphony: Distributed Hashing in a Small World. In Proceedings of the Fourth USENIX Symposium on Internet Technologies Systems (USITS), Seattle, WA, USA, 2003, 3: 127-140.

[62] Dahlia Malkhi, Moni Naor, David Ratajczak. Viceroy: A Scalable Dynamic Emulation of the Butter In Proceedings of the Twenty-First ACM Symposium on Principles of Distributed Computing (PODC), Monterey, CA, USA, 2002, 7: 183-192.

[63] Kaashoek M F, Karger D R. Koorde: A Simple Degree-optimal Distributed Hash Table. In Proceedings of the Second International Workshop on Peer-to-Peer Systems (IPTPS), Berkeley, CA, USA, 2003, 2.

[64] Karl Aberer, Philippe Cudré-Mauroux, Anwitaman Datta, Zoran Despotovic, Manfred Hauswirth, Magdalena Punceva, Roman Schmidt. P-Grid: A Self-organizing Structured P2P System. ACM SIGMOD Record, 2003, 9, 32(3).

[65] Ratnasamy S, Shenker S, Stoica I. Routing Algorithms for DHTs: Some Open Questions. Electronic Proceedings for the 1st International Workshop on Peer-to-Peer Systems (IPTPS'02).

[66] Considine J,Florio T. Scalable peer-to-peer indexing with constant state. Tech. rep. ,CS Department, Boston University,2002,9.

[67] Xu J,Kumar A,Yu X. On the Fundamental Tradeoffs between Routing Table Size Network Diameter in Peer-to-Peer Networks,IEEE Journal on Selected Areas in Communications,vol 22,no 1,2004,1: 151-163.

[68] Hu J F, Li M, Zheng W M, et al. SmartBoa: Constructing P2P Overlay Network in the Heterogeneous Internet Using Irregular Routing Tables. Proceedings of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS'04),2004,2.

[69] Sergio Marti, Prasanna Ganesan, Hector Garcia-Molina. DHT Routing Using Social Links, 3rd International Workshop on Peer-to-Peer Systems,2004.

[70] Moni Naor,Udi Wieder. Know thy neighbor's neighbor: Better routing for skip-graphs small worlds. Proceedings of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS'04),2004,2.

[71] Gummadi K,Gummadi R,Gribble S,Ratnasamy S,Schenker S,Stoica I. The impact of DHT routing geometry on resilience proximity. In Proc. of SIGCOMM, Karlsruhe, Germany, ACM, 2003, 9: 381-394.

[72] Wang S,Xuan D,Zhao W. On Resilience of Structured Peer-to-Peer Systems,in Proceedings of IEEE Global Communications Conference (GLOBECOM 2003) - General Conference,2003,12.

[73] Liben-Nowell David, Balakrishnan Hari, Karger David. Analysis of the Evolution of Peer-to-Peer Systems. ACM Conf. on Principles of Distributed Computing (PODC),Monterey,CA,2002,7.

[74] Nicholas J A Harvey, Michael B Jones, Marvin Theimer, Alec Wolman. Efficient Recovery From Organizational Disconnects in SkipNet. In Proc. IPTPS,Berkeley,CA,USA,2003,2.

[75] Chen R,Yeager W. Poblano: A distributed trust model for P2P networks. Technical Report,TR-I4-02-08,Palo Alto: Sun Microsystem,2002.

[76] Rao Ananth,Karthik Lakshminarayanan,Sonesh Surana,Karp Richard,Ion Stoica. Load Balancing in Structured P2P Systems. In Proc. IPTPS,Berkeley,CA,USA,February 2003.

[77] Karger D R,Ruhl Ma. New Algorithms for Load Balancing in Peer-to-Peer Systems. IRIS Student Workshop (ISW'03) Cambridge,MA,2003,8.

[78] Karger D R,Ruhl M. Simple Efficient Load Balancing Algorithms for Peer-to-Peer Systems. In ACM Symposium on Parallelism in Algorithms Architectures,2004,6.

[79] Byers John,Considine Jeffrey,Mitzenmacher Michael. Simple Load Balancing for Distributed Hash Tables. In Proceedings of the Second International Workshop on Peer-to-Peer Systems (IPTPS), Berkeley,CA,USA,2003,2.

[80] Xu Zhichen, Zhang Zheng. Building Low-maintenance Expressways for P2P Systems. Internet Systems Storage Laboratory,HP Laboratories Palo Alto,HPL-2002-41,2002,3(1).

[81] Xu Zhichen,Tang Chunqiang,Zhang Zheng. Building Topology-Aware Overlays using Global Soft-State. In ICDCS'03,2003,5.

[82] Garcés-Erice L,Ross K W,Biersack E W,Felber P A,Urvoy-Keller G. Topology-Centric Look-Up Service,In Proceedings of COST264/ACM Fifth International Workshop on Networked Group Communications (NGC),Munich,Germany,2003,9: 58-69.

[83] Freedman Michael J. Mazières David. Sloppy hashing self-organizing clusters. In Proceedings of the Second International Workshop on Peer-to-Peer Systems (IPTPS),Berkeley,CA,USA,2003,2.

[84] Xu Z Y,Min R,Hu Y M,HIERAS:A DHT-Based Hierarchical Peer-to-Peer Routing Algorithm,in Proceedings of the 2003 International Conference on Parallel Processing (ICPP'03), Kaosiung, Taiwan,2003,10: 187-194.

[85] Castro M,Druschel P, Hu Y C, Rowstron A. Topology-aware routing in structured peer-to-peer

overlay networks. MSR-TR-2002-82, September 2002. Available at ftp://ftp. research. microsoft. com/pub/tr/tr-2002-82. pdf.
[86] Castro M, Druschel P, Hu Y C, Rowstron A. Exploiting network proximity in peer-to-peer overlay networks. Tech. Rep. MSR-TR 2002-82, Microsoft, 2002.
[87] Zhao B, Duan Y, et al. Brocade: Lmark routing on overlay networks. In: Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS'02), Cambridge, MA, USA, 2002, 3.
[88] Ratnasamy S, Hley M, Karp R. Shenker S. Topologically-aware Overlay construction server selection. In Proc. INFOCOM'02, 2002.
[89] Saroiu S, Gummadi P K, Gribble S D. A measurement study of peer-to-peer le sharing systems. In Proceedings of Multimedia Computing Networking 2002 (MMCN), San Jose, CA, USA, 2002, 1.
[90] Zhu Y M, Wang H H, Hu Y M. A Super-Peer Based Lookup in Structured Peer-to-Peer Systems. Appears in Proceedings of the 16th International Conference on Parallel Distributed Computing Systems (PDCS'03). Reno, Nevada, 2003, 8.
[91] Mizrak A, Cheng Y, Kumar V, Savage S. Structured Superpeers: Leveraging Heterogeneity to Provide Constant-Time Lookup, Proceedings of the IEEE Workshop on Internet Applications, San Jose, CA, 2003, 6.
[92] Sean Rhea, Dennis Geels, Timothy Roscoe, John Kubiatowicz. Hling Churn in a DHT, in Proceedings of the USENIX Annual Technical Conference, 2004, 6.
[93] Bamboo. http://bamboo-dht. org.
[94] Li J, Stribling J, Gil T M, Morris R, Kaashoek F. Comparing the performance of distributed hash tables under churn. In IPTPS, 2004, 2.
[95] Christin N. Chuang J. On the cost of participating in a peer-to-peer network. Proceedings of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS'04), 2004, 2.
[96] Xu Z Y, Min R, Hu Y M. Reducing Maintenance Overhead in DHT Based Peer-to-Peer Algorithms. pp. 218-219, Linkpings, Sweden, 2003, 9.
[97] Mahajan R, Castro M, Rowstron A. Controlling the cost of reliability in peer-to-peer overlays. In IPTPS'03, 2003, 2.
[98] Rodrigo Rodrigues, Charles Blake. When Multi-Hop Peer-to-Peer Routing Matters. Proceedings of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS'04), 2004, 2.
[99] Prasanna Ganesan, Krishna Gummadi, Hector Garcia-Molina. Canon in G Major: Designing DHTs with Hierarchical Structure, 24th International Conference on Distributed Computing Systems (ICDCS'04), March 24 - 26, 2004. Hachioji, Tokyo, Japan.
[100] Bujor Silaghi, Bobby Bhattacharjee, Pete Keleher. Query Routing in the TerraDir Distributed Directory. In Victor Firoiu Zhi-Li Zhang, editors, Proceedings of the SPIE ITCOM 2002, volume 4868 of SPIE, Boston, MA, USA, 2002, 8: 299-309.
[101] Garcés-Erice L, EBiersack W, Felber P A, Ross K W, Urvoy-Keller G. Hierarchical Peer-to-peer Systems. Proceedings of ACM/IFIP International Conference on Parallel Distributed Computing (Euro-Par), 2003.
[102] Jeffrey Considine. Cluster-based Optimizations for Distributed Hash Tables, 2002. http://cs-www. bu. edu/ftp/fs/techreports/pdf/2002-031-DHT-cluster-optimization. pdf.
[103] Ganesan Prasanna, Sun Qixiang, Garcia-Molina Hector. YAPPERS: A Peer-to-Peer Lookup Service over Arbitrary Topology. In Proceedings of the Twenty-Second Annual Joint Conference of the IEEE Computer Communications Societies (INFOCOM), San Francisco, CA, USA, 2003, 3.
[104] Gupta I, Birman K, Linga P, Demers A, Renesse R V. Kelips *: building an efficient stable P2P DHT through increased memory background overhead. In Proceedings of the Second International

Workshop on Peer-to-Peer Systems (IPTPS), Berkeley, CA, USA, 2003, 2.

[105] Kempe David, Jon M Kleinberg, Alan J Demers. Spatial Gossip Resource Location Protocols. In Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing (STOC), Crete, Greece, 2001, 7: 163-172.

[106] Loo B T, Ryan H, Ion S, Joseph M Hellerstein. The Case for a Hybrid P2P Search Infrastructure. Proceedings of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS'04), 2004, 2.

[107] Yang B, Garcia-Molina H. Improving search in peer-to-peer systems. In Proc. of the 22nd International Conference on Distributed Computing Systems, Vienna, Austria, July 2002.

[108] Matthew Harren, Joseph M Hellerstein, Ryan Huebsch, Boon Thau Loo, Scott Shenker, Ion Stoica. Complex Queries in DHT-Based Peer-to-Peer Networks. In Peter Druschel, Frans Kaashoek, Antony Rowstron, editors, Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS), volume 2429 of Lecture Notes in Computer Science, Cambridge, MA, USA, 2002, 3: 242-250.

[109] Reynolds P, Vahdat A. Efficient Peer-to-Peer Keyword Searching. In Unpublished Manuscript, 2002, 6.

[110] Bloom B H. Space/Time Tradeoffs in Hash Coding with Allowable Errors. Communications of the ACM, 1970, 7, 13(7): 422-426.

[111] Bobby Bhattacharjee, Sudarshan Chawathe, Vijay Gopalakrishnan, Pete Keleher, Bujor Silaghi. Efficient peer-to-peer searches using resultcaching. in The 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03), 2003, 2.

[112] Bryce Wilcox-O'Hearn. Experiences deploying a large-scale emergent network. In Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS'02), Cambridge, MA, 2002, 3.

[113] Sun Qixiang, Garcia-Molina, Hector. Partial Lookup Services (Extended Version), Technical Report. Stanford University, 2002.

[114] Ozgur D Sahin, Abhishek Gupta, Divyakant Agrawal, Amr El Abbadi. A Peer-to-Peer Framework for Caching Range Queries, ICDE 2004.

[115] Sahin O, Gupta A, Agrawal D, Abbadi A. Query processing over peer-to-peer data sharing systems. Technical Report UCSB/CSD-2002-28, University of California at Santa Barbara, 2002.

[116] Gupta A, Agrawal D, Abbadi A E. Approximate Range Selection Queries In Peer-to-Peer Systems. In Proceedings of the First Biennial Conference on Innovative Data Systems Research (CIDR), Asilomar, CA, USA, 2003, 1: 141-151.

[117] Nathan Linial, Ori Sasson. Non-Expansive Hashing. In Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing (STOC), Philadelphia, PA, USA, 1996, 5: 509-518.

[118] Joseph S. P2P MetaData Search Layers. Second International Workshop on Agents Peer-to-Peer Computing AP2PC 2003.

[119] Zhang Z, Shi S M, Zhu J. SOMO: self-organized metadata overlay for resource management in P2P DHT. In Proceedings of the Second International Workshop on Peer-to-Peer Systems (IPTPS), Berkeley, CA, USA, 2003, 2.

[120] Crespo H, Garcia-Molina. Routing Indices for Peer-to-Peer Systems. In ICDCS, 2002, 7.

CHAPTER 3

第3章 对等网络拓扑及优化

3.1 对等网络的拓扑构造

3.1.1 引言

DHT-P2P 系统具有良好的特性是基于它的结构化拓扑。然而,由于将结点和数据文件紧密布置在一个结构化拓扑中,DHT-P2P 系统敏感于结构的动态变化。DHT-P2P 系统通过每个结点维护一个局部的路由表来实现全局路由和定位,通常对于 n 个结点的网络,对于每个结点的加入和离开,都需要 $O(\log^2 n)$的消息来维护正确的 P2P 网络拓扑结构[1]。当网络动态程序加大时,拓扑维护开销上升,从而在一定程度上限制了 DHT-P2P 系统的可扩展性。而事实上,对于 overnet 上的实验结果[2]表明几乎有 50% 的结点的会话时间小于一个小时,这说明在 DHT-P2P 系统中不得不在拓扑维护上增大开销。从这一点上而言,DHT-P2P 系统比非 DHT-P2P 系统在拓扑维护受到更直接的挑战。DHT-P2P 系统面临的一个基本问题是如何减少这些开销。

回答这个问题的根本出发点是:将参与 DHT-P2P 系统的结点区分成两类,一类结点具有足够稳定性,称为稳定结点,另一类结点并不是很稳定,具有较强波动性,称为自由结点。事实上,我们利用的是结点的会话异构特性[3]。基于此,我们设计了会话异构拓扑模型(Session Heterogeneity Topology,SHT)。SHT 模型是基于 DHT 的模型重构,将 DHT 的构造变成由稳定结点组成,从而降低了网络动态对 DHT 系统的影响,有力控制了 DHT 拓扑的维护开销。

3.1.2 SHT 模型

Stefan Saroiu 等用实验方法研究了 Napster 和 Gnutella 中的结点会话分布[3],结果如图 3-1 和图 3-2 所示。很明显地,有 50%的结点的会话时间小

于一个小时，这说明在 P2P 结点间存在很大的会话时间差异。

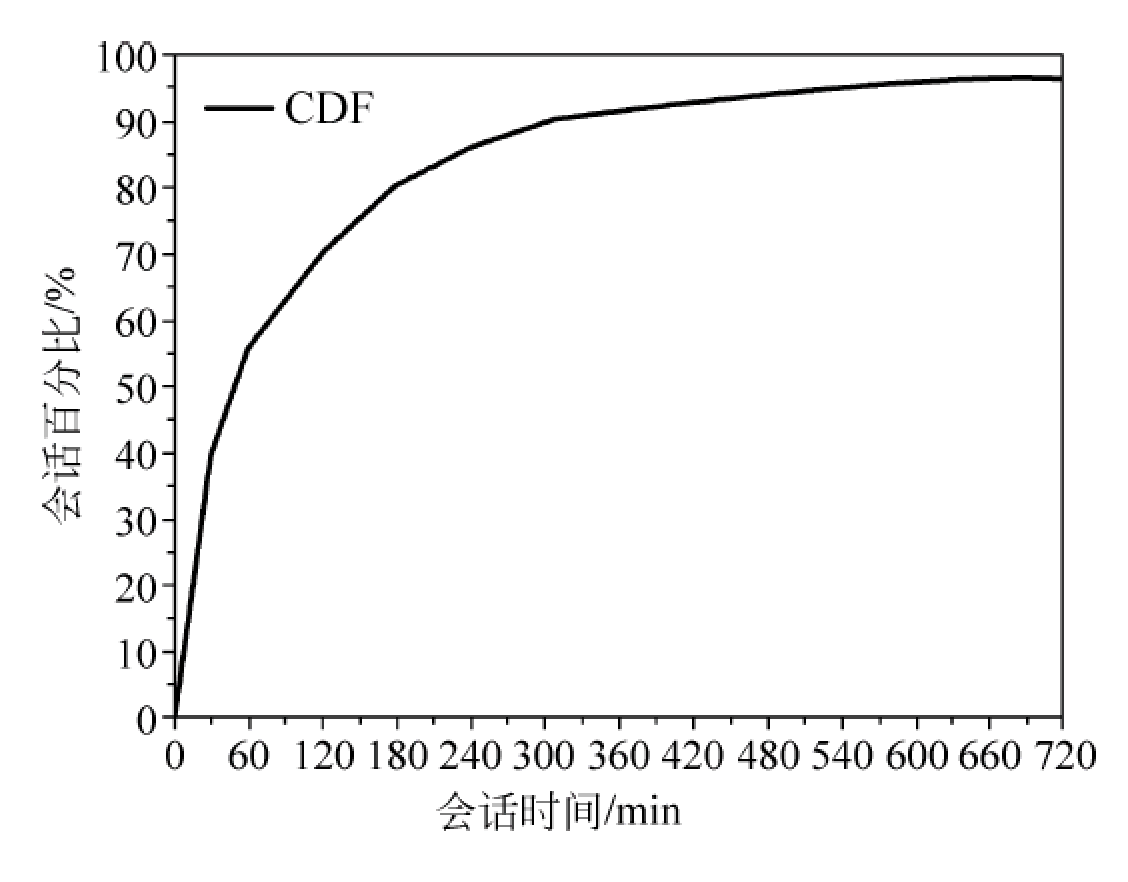


图 3-1 P2P 网络中的 Session 分布(CDF 曲线)

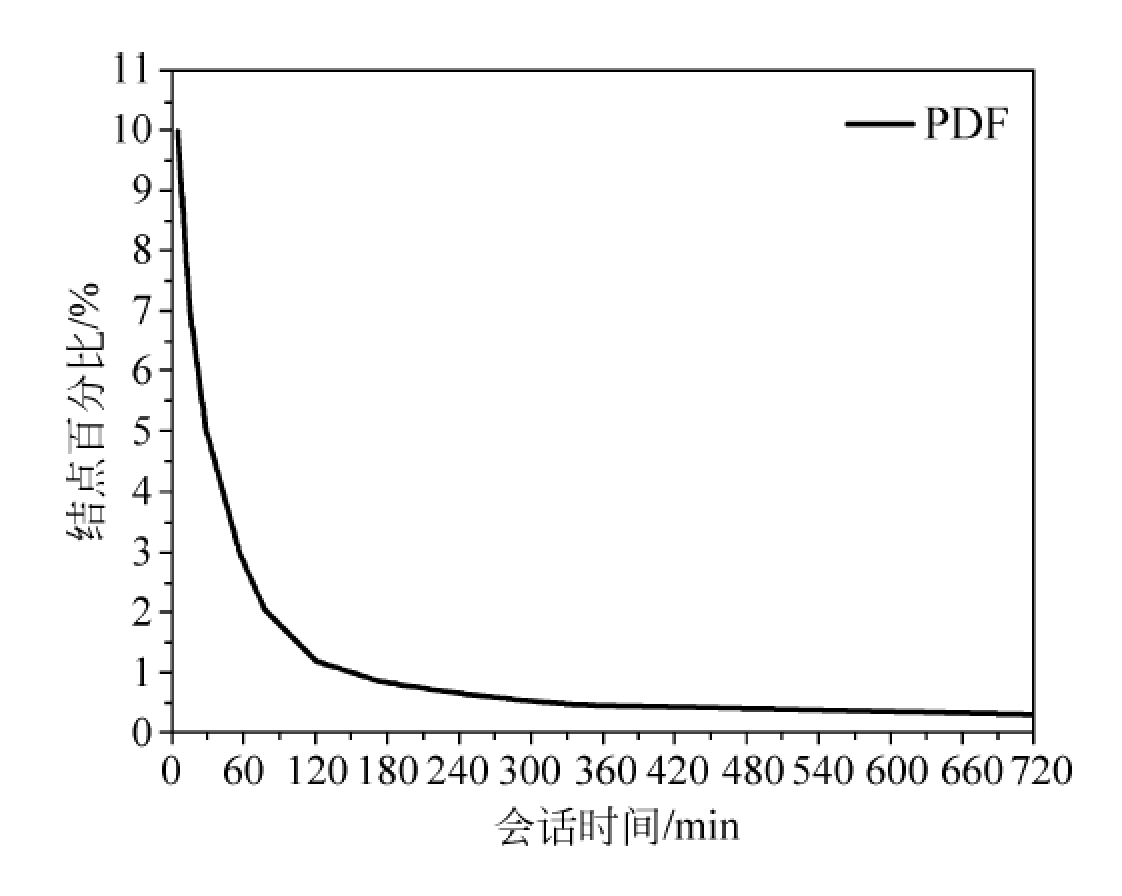


图 3-2 P2P 网络中的 Session 分布(PDF 曲线)

图 3-1 是 Session CDF (Cumulative Distribution Function，累积分配函数)曲线，图 3-2 是 Session PDF (Probability Density Function，概率密度函数)曲线。可以很清楚地观察到以下特点：

(1) 会话时间存在极大的差异性，这意味着对等结点并不是真正意义上的对等。

(2) 会话时间在结点间不均匀分布。当许多结点的会话时间非常短时，仅有一些结点的会话时间很长，似乎趋向于较高的动态性。

这个观察结果可以解释为什么结构化 P2P 系统需要很大的维护开销。许多会话时间很短的结点将不断频繁地加入，离开 P2P 网络，以至于扰乱了网络拓扑的结构，快速地增加了结构化 P2P 拓扑的维护开销。考虑到这个事实，一个直觉的想法就是剔除这些结点参与 DHT 拓扑构造。因此，我们利用了会话时间的极大差异性来构造 SHT 模型，让稳定的结点构造起 DHT 环(将所有由 DHT 技术构造起来的拓扑如 Chord、Pastry、CAN 和 Tapestry 归纳为环空间结构，称为 DHT 环)，而其他的自由结点将在 DHT 环以外。这就意味着动态的不稳定的结点被集成簇留在 DHT 环以外，这样就降低了 DHT 环的动态改变程度。由于 DHT 环由这些稳定的结点组成，DHT 环的维护开销就能被大大降低，从而整个拓扑维护开销也就大为减少。在这里，结点的稳定性是由在线会话时间来评估的，会话时间最长的结点将被选到 DHT 环上去。可能在选择结点的时候，还有更多的因素如 CPU 处理能力、带宽等，会被考虑权衡，然而会话时间仍是一个影响维护开销的主导因素。因而，我们将这些因素的折中问题留给未来的工作考虑，而在这里仅关注会话时间这一支配性因素。

DHT 环上的结点称为父结点，而环外以环上一个父结点为中心的结点称为子结点。一个父结点和它的子结点集称为一个簇。对于一个含有 n 个结点的网络，用 m 来限制每个父结点的子结点个数。对此，期望在一个合理的 m 值之后，可以采用聚簇策略挑选出全局稳定性能足够的结点来成为父结点，以形成一个稳定的 DHT 环。因此，一个通用的 SHT 拓扑结构是由簇集合 $S=\{C_1, C_2, C_3, \cdots, C_i, \cdots\}$ 构成的，其中 C_i 是一个簇，并且 $i\in[1,n]$。对于 $\forall C_i \in S$，都有 $C_i=\{f, s_1, s_2, s_3, \cdots, s_i, \cdots\}$，其中 f 是父结点，s_i 是子结点，$i\in[1,m]$。所有的父结点组成了一个 DHT 环，有父结点集 $D=\{f_1, f_2, f_3, \cdots, f_i, \cdots\}$，$i\in$

$[1,n]$。SHT 模型核心技术就是要从簇中选出一个稳定的结点到 DHT 环上去作为父结点。所以每个父结点都将维护子结点列表 L，其中包含了所有子结点的注册信息(即结点 ID、IP、会话时间)。父结点是从 L 中挑选出来的，并通过一种渐近方式来形成稳定的 DHT 环。

图 3-3 描述了 SHT 模型的拓扑结构，在该图中，DHT 环上实心小环代表了父结点，而连接在这些父结点上的方块代表着子结点。

图 3-3　SHT 模型的拓扑结构

1. 结点加入

对于一个任意的结点 a，其加入到网络中的过程描述如下：

(1) 结点 a 与一个已知的引导结点 I(Introducer)联系。然后通过结点 I 发送加入请求以把自己加入到一随机簇 C 中。

(2) 如果簇 C 的大小超过 m，那么簇 C 就分裂成两个簇，并根据新簇选出新的父结点。①从簇 C 的所有子结点中选出会话时间最长结点 b 作为新簇父结点加入 DHT 环，命名此新簇名为 C'，此时它的子结点数为 0，并从簇 C 中删除结点 b。②从簇 C 中随机移动 $\left\lfloor \frac{m}{2}-1 \right\rfloor$ 个子结点到簇 C'。

2. 结点离开

在 SHT 模型中，对于不同类型的结点离开，采用不同的处理办法。当一个子结点离开的时候，发送一个通知给父结点，告知更新了的子结点列表。当父结点离开的时候，从子结点中挑选稳定的结点作为一个新的父结点，然后这个父结点就被加入到 DHT 环上。对子结点列表的动态维护，以及父结点的选择过程将在下面描述。

3. 渐近稳定的 DHT 环

尽管结点的加入和离开是非常简单的，但是它能用一个进化的方法来渐近形成稳定的 DHT 环。这里说的进化是指拓扑结构逐渐朝着期望的方向改变。有以下两种情况会从簇中选择新父结点：簇的大小超过了 m，簇中父结点的异常事件(如发生掉电事故)。每次选择均保证父结点为簇中局部的最稳定结点，从而期望父结点在全局是最稳定的结点。父结点的选择对于一个稳定的 DHT 环是十分关键的，起决定性作用。

为了形成一个稳定的 DHT 环需回答以下两个问题：

(1) 怎样使得父结点的选择既可行又可靠？

(2) 这些父结点是否能够使得 DHT 环达到想要的稳定状态？

首先，可通过维护子结点列表来解决第一个问题。对于一个簇来说，父结点维护着可能成为下一次父结点候选者的子结点有序列表。簇中一个子结点存活的时间越长，即它的会话时间越长，它就越有可能被选为父结点，在列表中位置就越靠前。而且，为了应付一些如掉电、断网等异常失效情况，需要将这个子结点列表周期性地发送给簇上的每个子结点。当一个父结点失效或断开的时候，列表中会话时间最长的结点就成为父结点，并加入到 DHT 环上。通过这种方法，父结点的选择就将是可行又可靠的了。

接下来来考虑第二个问题。DHT 环的一个理想的稳定状态定义如下：

定义 3-1 令 *node*.*session* 为结点的会话时间，令 U 为所有子结点的集合。对于一个含有 n 个结点、簇的大小是 m 的网络来说，如果它满足两个条件：

（1）$D=\{f_1,f_2,\cdots,f_v\}$，$v=\lceil n/(m+1)\rceil$；

（2）$\forall f\in D, \forall u\in U, f.session\geqslant u.session$。

即为 DHT 环的稳定状态。

下面来说明一下如何能够渐近满足这两个稳定条件。

因为聚簇技术采用及均匀性分割，最终将在 DHT 环上留下 v 个父结点。因此，它满足条件(1)。让 F 是所有环上父结点集，即 $F=D=\{f_1,f_2,\cdots,f_v\}$，并让它们对应的簇集为 C，即 $C=\{C_1,C_2,\cdots,C_v\}$。因为对于 $\forall f_i\in F$，根据选择策略，$f_i.session$ 将是相应簇 C_i 的最大值。似乎这样就满足了条件(2)了，但事实上，这并没有满足条件(2)。因为这时的 $f_i.session$ 只能表明它在局部即本簇内是最大，并不能够表明它在全局即所有子结点集是最大。因此，采用了一个随机算法来持续进行调整，称为渐进的稳定化。

函数 adjust(c_i) 运行在每个父结点（c_i 即为父结点所在代表的簇），并以一定时间周期调用。它随机检测其他簇，每次检测可能不满足条件(2)的簇，从而以一种渐近方式调整 DHT 环以达到条件(2)的要求，其中 $c_i\in C$。

随机调整算法的伪代码如下：

```
   adjust(ci){
1  randomly select a cluster c,where c∈C⋂c≠ci
2  select node t whose session is the maximum of the son node set in c
3  if (t.session > ci.f.session)
4      c = c - {t} + {ci.f}      // adjust cluster c
5      ci.f = t                  // update the father of cluster ci with node t
   }
```

上面有标号的第 1～3 行是根据当前簇 c_i 中的父结点，随机选择网络中另外一个簇 c，并判断是否存在簇 c 中某子结点 t 的会话时间比当前簇 c_i 的父结点会话时间更长。如果存在这种情况，那么就执行有标号的第 4～5 行，即将簇 c 的子结点 t 与当前簇 c_i 的父结点交换。

4. 文档发布和搜索

对每个结点来说，发布和搜索文档是非常重要的。DHT 模型提供高效率的文档发布和搜索，这是以在一定程度牺牲其拓扑灵活性为代价的。SHT 要保留 DHT 的优点，同时还要克服它的缺点，所以 SHT 使用稳定的 DHT 环来存储索引元数据和搜索文档。任意一个结点，无论它是父结点还是子结点，均能够发布①和搜索文档，但是文档元数据（文档索引数据）的存储、查询和路由必须基于 DHT 环，这样才能利用 DHT 拓扑构造的优点。

对于一个父结点，它位于 DHT 环上，所以它的发布和搜索文档的操作跟 DHT 算法一样。然而，子结点的操作则需要用到一个代理结点，也就是它们的父结点。当子结点发布文档时，发布请求将被发送给它的父结点，父结点在 DHT 环定位并存储发布文档元数据。子结点则无须知道元数据是怎样存储的，存储在哪里。同样地，子结点提交查询请求给它的父

① 发布文档就是指提供文档的索引元数据，以便文档可被搜索。

结点，然后父结点在 DHT 环上用特定的 DHT 算法执行查询，并把最终的结果传回给它。

为了避免无效的元数据信息存储在 DHT 环上影响到搜索效率，采用了软状态(Soft State)更新技术来发布文档。文档的根结点通过周期性地发布该文档来刷新元数据信息。存储元数据信息的结点将定期删除一些过时的元数据。

这种文档发布和搜索基于一个相对稳定的 DHT 环，因而效率得到提高，数据的可用性也将得到大大改善。

5. 分析

簇的大小 m 对于 SHT 模型是一个关键参数。它决定性地影响 DHT 环的稳定性和维护开销的大小。下面将分析 m 是怎样影响稳定性和维护开销的。分析基于上面所描述的稳定 DHT 环，并且会话时间的分布来自参考文献[3]中的真实跟踪数据(trace)。以下所指的时间如不加注明皆指会话时间。我们考虑的是一种均衡性网络，结点以泊松率加入与退出系统，但系统内结点总数保持不变。

定义 3-2　令 C 为结点会话时间的 CDF，令 P 为结点会话时间的 PDF，令 T 为时间。显然可以得到

$$P(t) = \lim_{\Delta t \to 0} \frac{C(t+\Delta t) - C(t)}{\Delta t}, \quad t \in T \tag{3-1}$$

定义 3-3　令 G 为所有结点集。如果簇的大小是 m，那么 G 就被分成了 2 个子集：G_{circle} 和 G_{outer}，其中 $G_{circle} = \{x \mid x.session \geqslant t_m, x \in G\}$，$G_{outer} = \{x \mid x.session < t_m, x \in G\}$。$t_m$ 是分割这两个集合的会话时间点。

事实上，观察图 3-1，可以知道，m 是 DHT 环上的结点数和 DHT 环外的结点数的比值，即

$$m = \frac{C(t_m)}{1 - C(t_m)}, \quad C(t_m) = \frac{m}{m+1} \tag{3-2}$$

下面来看看故障率和 m 之间的关系。

定义 3-4　结点的故障率是指在 Δt 时间内结点动荡(加入或者离开)数目，取值为 Δt 趋于 0。结点在时间 t 的故障率记为 $f(t)$，M 为结点总数，则有

$$f(t) = \lim_{\Delta t \to 0}\left(\frac{1}{Dt} \times \frac{M \times C(t+\Delta t) - M \times C(t)}{M \times C(t)}\right) = \frac{P(t)}{C(t)} \tag{3-3}$$

通过式(3-2)和式(3-3)，采用 trace 数据中的会话分布，可以描绘出图 3-4。图 3-4(a)表明 $f(t)$ 随时间而递减，图 3-4(b)表明 $f(t_m)$ 将随 m 增大而递减。事实上，有如下定理 3-1：

定理 3-1　系统内的结点故障率将随着其会话时间增长严格单调递减。

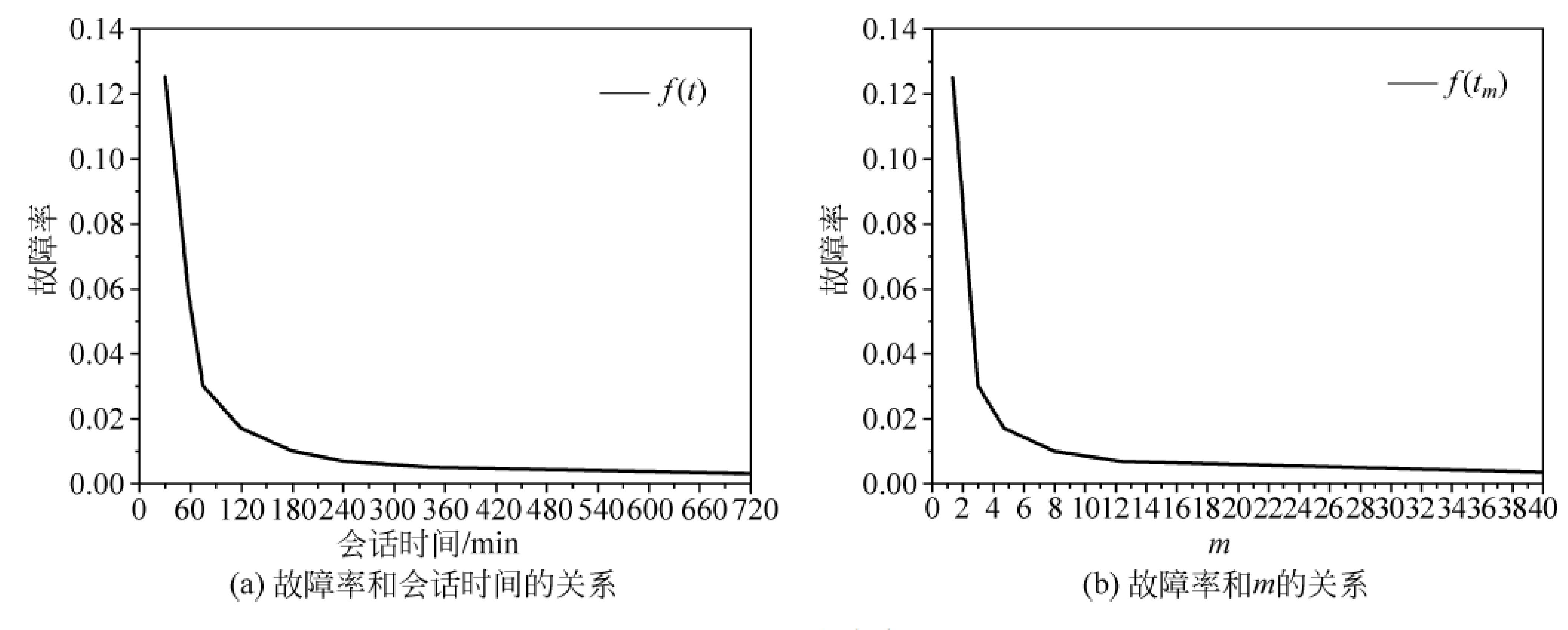
证明：

因为

$$f(t) = \frac{P(t)}{C(t)}, \quad P(t) = C'(t)$$

故

$$\begin{aligned} & f'(t) = \frac{P'(t)C(t) - P(t)C'(t)}{C^2(t)} = \frac{P'(t)C(t) - P^2(t)}{C^2(t)} \\ & P'(t) < 0, \quad 0 \leqslant C(t) \leqslant 1 \\ & f'(t) < 0 \end{aligned} \tag{3-4}$$



(a) 故障率和会话时间的关系　(b) 故障率和m的关系

图 3-4　故障率

所以，$f(t)$是 t 的严格单调递减函数，即结点故障率随着会话时间增长严格单调递减。证毕。

定义 3-5　系统的稳定性是根据系统结点的平均故障率来衡量的。若 S 代表平均故障率，则有

$$S=\frac{1}{M}\int_0^t(M\times P(t)\times f(t))\mathrm{d}t=\int_0^t\frac{P^2(t)}{C(t)}\mathrm{d}t \tag{3-5}$$

若 S_{circle} 为 DHT 环的稳定性，则有

$$S_{\mathrm{circle}}=\int_{t_m}^t\frac{P^2(t)}{C(t)}\mathrm{d}t \tag{3-6}$$

S 或 S_{circle} 越小，说明故障率越低，从而表明系统稳定性更高。

引理 3-1　S_{circle}将随 m 加大而减小，即 DHT 环稳定性随 m 加大而增强。

证明：从式(3-2)易知 t_m 将随 m 值增加而增长(注意 C 是严格单调增函数)。因为式(3-6)中 S_{circle}是区域$[t_m,t]$的积分，所以 S_{circle}将根据 t_m 的增长而减少。也就是说，m 增长导致 S_{circle}减少。证毕。

定义 3-6　结点进出率 R 是指单位时间内加入或者离开网络的平均结点数。

定义 3-7　维护开销 F 是指维护叠加网络结构的开销。F_{DHT} 为 DHT 模型拓扑维护开销，F_{SHT} 为 SHT 模型拓扑维护开销。

结点进出率也是系统稳定性的一种度量。R 越小，说明单位时间内进出网络的结点数越少，即网络波动性小，也就是系统稳定性增强。

对于基于 DHT 模型有 n 个结点的网络，假定结点的进出率是 R_{DHT}。在采用 SHT 进行拓扑进化后，则 R_{SHT}实际分成两部分：一部分是 DHT 环上的结点进出率 R_{circle}，另一部分是环外的结点进出率 R_{outer}。

引理 3-2　R_{circle}将随 m 增大而减小，即 DHT 环结点进出率随 m 增大而减少。

证明：

让 T_{u} 代表单位时间，由定义 3-5、定义 3-4 和定义 3-6 可推出

$$R=S\times T_{\mathrm{u}} \tag{3-7}$$

即

$$R_{\mathrm{circle}}=S_{\mathrm{circle}}\times T_{\mathrm{u}} \tag{3-8}$$

$$R_{\text{outer}} = S_{\text{outer}} \times T_{\text{u}} \tag{3-9}$$

由引理 3-1 可知，S_{circle}将随 m 增大而减小，即 R_{circle}将同样随 m 增大而减小，即 DHT 环结点进出率随 m 增大而减少。证毕。

引理 3-3　结点进出率在 SHT 模型分割成两部分，且有 $R_{\text{DHT}} = R_{\text{circle}} + R_{\text{outer}}$。

证明：由定义 3-3 可知 $G = G_{\text{circle}} + G_{\text{outer}}$，并可得知 S_{circle}实际是 G_{circle}平均故障率的期望，S_{outer}实际是 G_{outer}平均故障率的期望。因此有

$$\left.\begin{aligned} &E(G) = E(G_{\text{circle}} + G_{\text{outer}}) = E(G_{\text{circle}}) + E(G_{\text{outer}}) \\ &S = S_{\text{circle}} + S_{\text{outer}} \\ &R = R_{\text{circle}} + R_{\text{outer}} \end{aligned}\right\} \tag{3-10}$$

又因为

$$R_{\text{DHT}} = R_{\text{SHT}} = R \tag{3-11}$$

所以

$$R_{\text{DHT}} = R_{\text{circle}} + R_{\text{outer}} \tag{3-12}$$

也就是说 DHT 模型经 SHT 模型进化后的结点进出率在 DHT 环内外分割，总和仍然保持不变。证毕。

定理 3-2　取 $m_t = \max\{m \mid R_{\text{circle}}(m) - R_{\text{circle}}(m+1) < e, m = 1,2,3,\cdots\}$后，维护的开销将得到一个近似的最小值，其中 e 是平滑阈值。且相比之于 DHT 模型，维护开销的减少值 H 近似为

$$\begin{aligned} H &= F_{\text{DHT}} - F_{\text{SHT}} \\ &= R_{\text{DHT}} \times T \times (O(\log^2 n) - O(m_t)) - R_{\text{circle}} \times T \times O(\log^2[n/(m_t+1)]) \\ &\approx (R_{\text{DHT}} - R_{\text{circle}}) \times T \times O(\log^2 n) \quad (m \ll n) \end{aligned} \tag{3-13}$$

证明：DHT 和 SHT 两模型的维护开销如下(其中 T 是运行时间)：

$$F_{\text{DHT}} = R_{\text{DHT}} \times T \times O(\log^2 n) \tag{3-14}$$

$$F_{\text{SHT}} = R_{\text{outer}} \times T \times O(m) + R_{\text{circle}} \times T \times \{O(\log^2[n/(m+1)]) + O(m)\} \tag{3-15}$$

由引理 3-3 及式(3-12)，对式(3-15)可简化为

$$F_{\text{SHT}} = R_{\text{DHT}} \times T \times O(m) + R_{\text{circle}} \times T \times O(\log^2[n/(m+1)]) \tag{3-16}$$

令

$$F_{\text{outer}}(m) = R_{\text{DHT}} \times T \times O(m) \tag{3-17}$$

$$F_{\text{circle}}(m) = R_{\text{circle}} \times T \times O(\log^2[n/(m+1)]) \tag{3-18}$$

则

$$F_{\text{SHT}} = F_{\text{outer}}(m) + F_{\text{circle}}(m) \tag{3-19}$$

由引理 3-2 可知，R_{circle}随簇 m 增加而减小。特别地，在 m 取定的一个值后，R_{circle}将急剧减小，并趋于平滑，这与会话分布相关(见图 3-4)。令 e 为平滑阈值，e 是一个正实数，使其小得足够反映 DHT 环的稳定度。假设

$$m_t = \max\{m \mid R_{\text{circle}}(m) - R_{\text{circle}}(m+1) < e, m = 1,2,3,\cdots\} \tag{3-20}$$

由 $m = m_t$，会得到 R_{circle}一个近似的最小值，这时候系统的维护开销 F_{SHT}在 m_t 值时达到近似优化。首先，因为再增大 m_t 对于环稳定性增加不多(e 值相当小)，由式(3-18)可知，此时环上结点的维护开销 $F_{\text{circle}}(m)$达到近似最小。此外，由于极端异构性，m_t 相对 n 仍然会是个可期望的小值，从而由式(3-17)可知，$F_{\text{outer}}(m)$在 m_t 值处近似最小。因此，由式(3-17)

可得 F_{SHT} 在 m_t 值时达到近似优化值。

令 H 为维护开销为近似最小时的减少值，则有如下表达式：

$$\begin{aligned} H &= F_{DHT} - F_{SHT} \\ &= R_{DHT} \times T \times [O(\log^2 n) - O(m_t)] - R_{circle} \times T \times O(\log^2[n/(m_t+1)]) \\ &\approx (R_{DHT} - R_{circle}) \times T \times O(\log^2 n) \quad (m \ll n) \end{aligned} \tag{3-21}$$

证毕。

由定理 3-2 可知，维护开销得到了较大的减少，并且近似最小维护开销的簇大小 m_t 可以通过 DHT 环上结点进出率 R_{circle} 的测量计算得到。式(3-21)表明了开销减少的本质在于将结点的进出率由 R_{DHT} 控制到一个稳定环的进出率 R_{circle}。

3.1.3 仿真实验

基于 3.1.2 节的描述设计，实现了一个 SHT 仿真原型。DHT 环是根据 Chord 协议构造的，并且采用了与 Chord 同样的机制(以 30s 为稳定间隔时间)，因此称为 SH-Chord。当然，对 DHT 环的构造也可基于 CAN、Pastry、Tapestry 等，然而，这并不改变 SHT 模型的根本效果。

下面介绍的实验由两个部分组成。首先，观察簇的大小 m 是怎样影响 DHT 环的稳定性(以结点进出率度量)，为设定拓扑优化提供重要的参数值 m_t；其次，观察使用 SHT 模型后对于维护开销的控制及数据可用性等性能的影响情况。

1. DHT 环稳定性和簇的大小的关系

这里在网络中加入了 20 000 个结点。为了得到一个现实的效果，会话时间的分布采用真实的跟踪数据[3]。为了观察到簇的大小是怎样影响 R_{circle} 的，从 1 到 40 改变簇的大小 m。由于所有的元数据信息都被发布并存储到 DHT 环上，搜索也是在 DHT 环内完成的，所以 DHT 环的稳定性体现了 SHT 模型的稳定性。以环的结点进出率 R_{circle} 来度量 DHT 环的稳定性。测量每分钟环上结点的进出率，并计算出平均值作为 R_{circle}。实验结果如图 3-5 所示。随着 m 的增加，R_{circle} 将首先急剧下降，但是在 m 到达 10 左右之后，就开始变得平滑。由定理 3-2 可知，m_t 近似为 10，这时的维护开销接近最小值。代替这种人工观察，也可以由初始设定的 e 值，调整 m 值，测量得到 R_{circle}，通过不断逼近的方法同样可以得到 m_t 值。

2. SHT 的性能

为了更好地观察 SHT 模型对于维护开销的控制和对数据可用性的影响，设计的实验基于优化的情况，即簇大小参数取测量优化值。实验设置参数如下：结点会话时间分布采用同样的跟踪数据，簇大小取测量值 m_t 为 10，20 000 个结点持续以每秒钟一个的速率加入到网络中，为了观察 SHT 模型是怎样影响维护开销的，这样的高速率是我们想要的动态网络环境。实验运行 24 小时，且在运行期间每 10 分钟，系统性能参数被快照一次。

1) 维护开销

整个系统范围内的每分钟收到的消息数来评估维护开销。在图 3-6 中可以看到，SH-Chord 的维护开销大约是每分钟 5000 条消息，但是 Chord 则是每分钟 20 万条消息。因为系统中有 20 000 个结点，所以，对于 SH-Chord 是平均每个结点每分钟 0.25 条消息，而对于

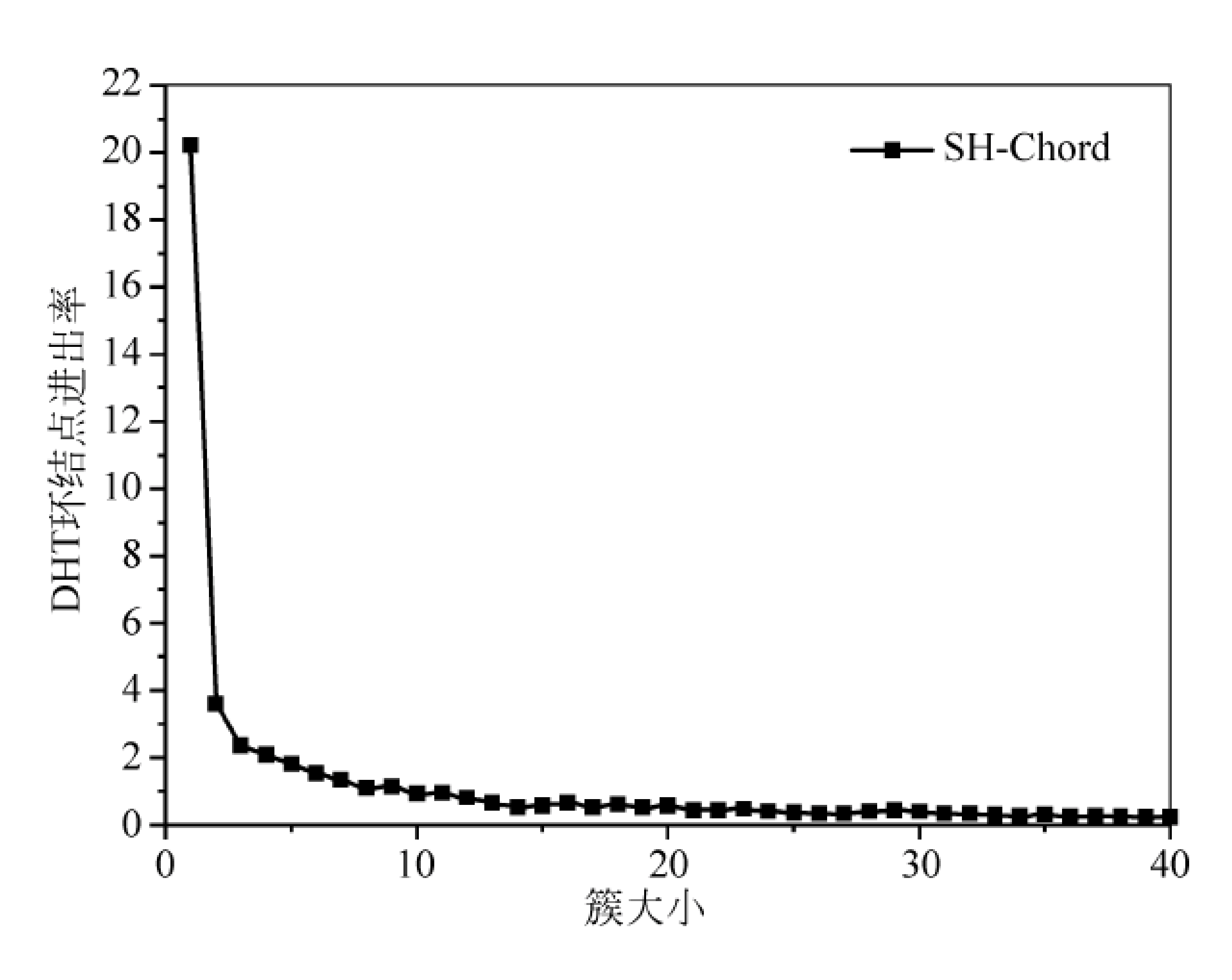


图 3-5　DHT 环结点进出率和簇大小的关系(簇大小变化从 1 到 40)

Chord 则是平均每个结点每分钟 10 条消息。这样,使用了 SHT 模型的簇技术后,维护开销被降低到只有原先 Chord 中的 2.5% 。这是依据定理 3-2,当簇大小为 m_t 时,DHT 环将会很稳定,因而大大减少了维护开销,使维护开销得到有力控制。

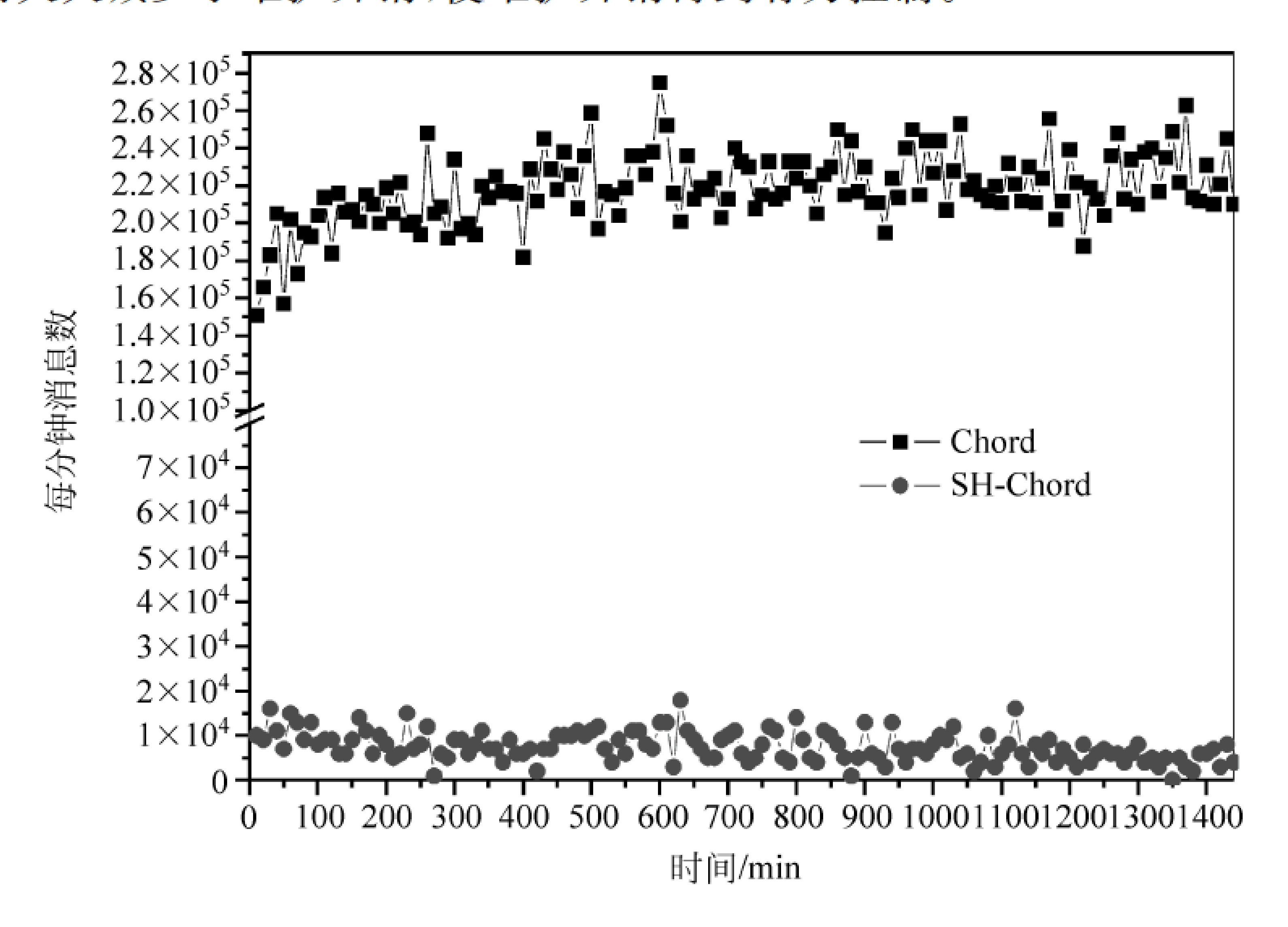
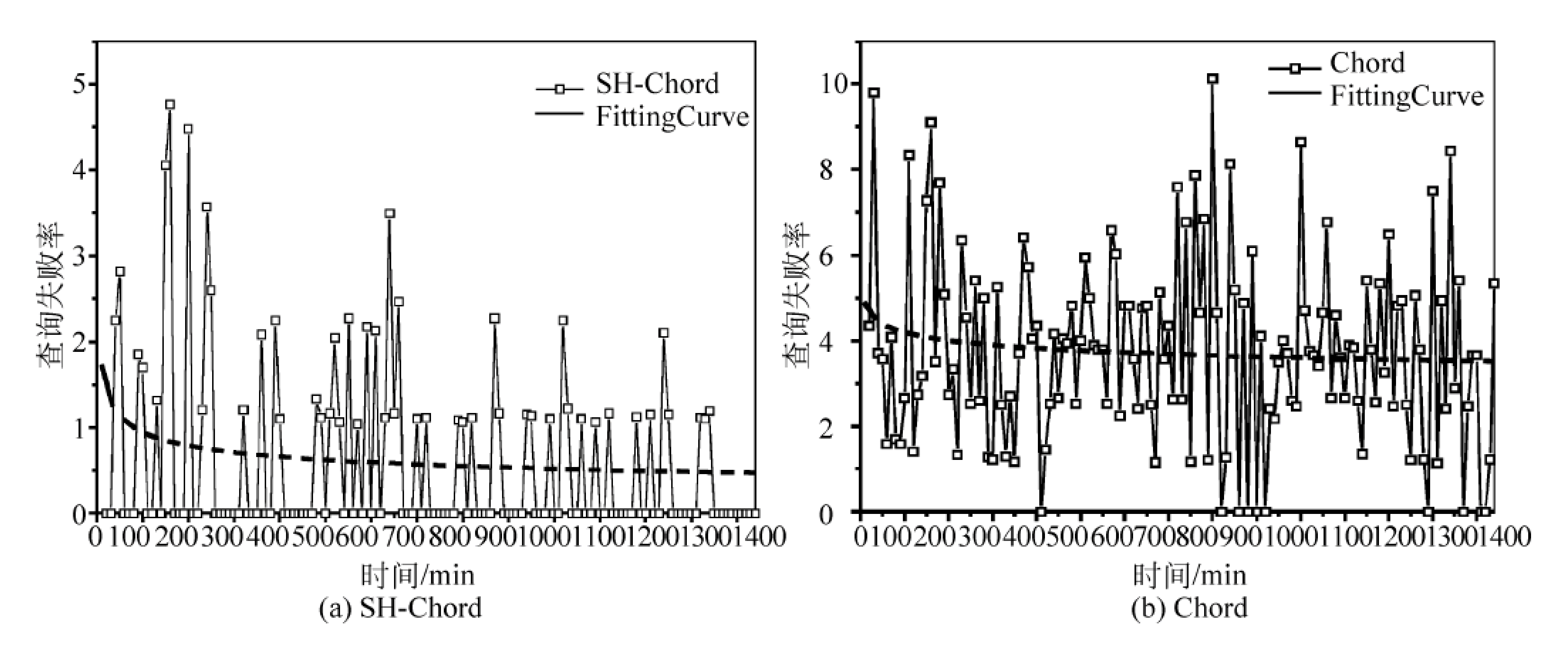


图 3-6　SH-Chord 和 Chord 之间的维护开销的比较

2）查询失败率

图 3-7 显示了 SH-Chord 和 Chord 相应的查询失败率及拟合曲线,其中点线是拟合曲线。可以观察到查询失败率随时间上下波动,这与在极度动态的网络中执行的查询的波动相一致 (每秒 1 个结点)。从拟合曲线中可以看到,SH-Chord 的查询失败率是 0.005,Chord 是 0.04。SH-Chord 失败率比 Chord 小了一个数量级,这是由于 SHT-Chord 中路由查询采用了渐近稳定的 DHT 环。实验结果表明,SHT 模型比 DHT 模型有着更好的数据可用性。



(a) SH-Chord (b) Chord

图 3-7　查询失败率和拟合线

3.1.4　相关工作

P2P 网络的高度的动态性和异构性已经被最近的实验观察所证实[2,4]。Ratnasamy 等[5]在 IPTPS02 会议论文中提出,DHT 路由方面的开放性问题,其中之一就是利用 P2P 系统中结点异构性来改善它的性能。当前研究从各种异构特性出发改进系统的性能,如利用带宽[3]、主机[6]、地理位置[7]等不同的异构特性。区别与这些研究,本书利用的会话异构,从另一角度挖掘出 P2P 网络中结点内在异构性,并基于此改进 DHT 拓扑。

DHT 拓扑维护问题在参考文献[4]中做了分析。Ledlie 等通过 Chord 实验指出,为了保证实际拓扑的有效,对等网络的参与结点至少要满足以下两个条件之一:①所有结点都具有足够的带宽支持每秒可运行稳定化过程多次;②所有结点都有足够长的会话时间。文献[3]的实验观察得出结点异构巨大的差异性,使得这两个条件的任何一个都不可满足。因而,DHT 拓扑必须充分考虑其拓扑维护问题。SHT 模型能够考虑结点间的异构差异性,从而挑选出满足①、②的强能力结点思路加入 DHT 拓扑环,从而能够较好解决这个问题。Liben-Nowell 等[8]分析了 P2P 网络的动态特性,给出了 Chord 拓扑维护的一个近似分析。Ratul Mahajan 等[9]针对 DHT 拓扑维护问题,提出了一种控制维护开销模型,其要点是通过自适应网络动态变化调节对等网络的关键参数以平缓动态影响。本质上它是一种被动方式,忽略了结点异构特性,当结点动态增强时,会有结点瓶颈问题而导致调节失效。与之相反,SHT 模型是一种主动控制网络动态的模型构造,它能够主动选择出稳定结点作为 DHT 环来并控制不稳定的动态结点,从而保障了环的稳定性。

SHT 模型充分利用了会话异构的反馈性能。尽管我们的分析和实验是基于参考文献[3]中的真实跟踪数据的,这是发现的最早的有关 P2P 系统中会话异构性的研究,但是我们的方法实际上可同样适用于会话异构性显著的一大类 P2P 系统。此外,我们也必须指出,SHT 模型可能并不是非常有效于会话异构并不明显的一类系统。但是,参考文献[3]发表之后的一系列实验研究,如参考文献[2,10],同样显示了当前正在使用的 P2P 系统所具有的强烈异构特性。因此,SHT 模型有着它现实意义上的重要性,给予现实 P2P 系统经过很小的修改从而能够更好适应动态网络,提高系统的健壮性和稳定性。

结构化 DHT-P2P 系统能够提供高效、可靠的服务，有着巨大的潜在应用前景。然而，现实应用中维护 DHT 拓扑，巨大的开销是不可避免的，这就限制了它的应用，尤其是在高度动态的环境下。上面通过 P2P 网络中的会话异构性，提出了一个创新的 SHT 模型来控制维护开销。SHT 模型使用了一个简单但是有效的簇技术。主要的技术优点有：

(1) 簇的管理方式是自然而演化的，管理开销很小。由于它不依赖于任何附加的前提条件，所以它可以直接应用于现有的 DHT 算法的改进。

(2) 利用了网络异构特性，即使簇的大小很小，它也能得到一个近似理想的效果，从而对簇的中心结点要求较低，具有良好的现实适用性。

仿真结果表明维护开销可以被极大地减少，而查询失败率也很大程度减小。因此，采用 SHT 模型在控制拓扑维护开销的同时，进一步提高了 DHT-P2P 系统的稳定性和数据可用性。

3.2　对等网络的路由优化

3.2.1　引言

路由是 P2P 系统的基础，提高路由性能是 P2P 系统的一个极关键问题，因而非常受到重视。现在正在使用的 DHT-P2P 系统，诸如 CAN、Chord、Tapestry、Pastry，路由性能各有它们的优点和局限性。CAN 的路由效率相对比较糟糕，特别是在低维的情况下。但它能使用低度的路由表来保持相对较低的维护开销。Chord、Tapestry、Pastry 有着较优的路由效率，却不得不忍受较高的维护开销。在负载较重的动态环境下，它甚至导致路由颠簸问题[11]。

下面将研究 DHT 路由的一种改进策略，并由此展示 DHT-P2P 系统中以较低的维护开销实现有效的资源定位的可行途径。研究工作是以 CAN 为基础，但略加变换后同样适用于其他的 DHT 路由算法。本节的主要创新是在结构化路由层上构造了小世界模式，并由此改进路由性能。

这里的工作主要受 Kleinberg 最近研究成果的启发[12]。Kleinberg 提出，如果将 n 个结点放入一个二维网格中并且每个结点都有一些短链和仅一条长链，采用贪婪路由能够以平均 $O(\log^2 n)$跳在任何一对结点间传送消息。Kleinberg 的理论对建立和改进 P2P 系统非常有用。目前已有一系列 Kleinberg 式的网络构造见参考文献[13，14]。在 CAN 中构建小世界是一个非常直觉的想法。因为 CAN 有着和 Kleinberg 模型相似的拓扑结构。然而，Kleinberg 模型是一个静态模型，它使用所有结点的全局信息来构建小世界现象。反过来，P2P 系统中结点的加入和离开是自由的，并且每个结点只拥有整个系统的一小部分信息。例如，CAN 结点是自由地加入和离开的，并且每个结点只知道它的 $2d$ 个邻居结点。因此，尝试概率缓存结点代替 Kleinberg 模型中的静态结构来模型小世界现象。这种方法的潜在收益是能够以一种极低廉的代价来改进路由性能。因为概率缓存技术在改进 CAN 的设计中非常重要，所以我们将此改进的 CAN 命名为 PCCAN(Probabilistic Cache-based CAN)。

3.2.2 小世界模型

小世界模型即大多数人都被许多条由熟人构成的短链连接着的理论，是由 Stanley Milgram[15]在 20 世纪 60 年代作为社会学问题首次提出的。可以用 Milgram 实验中一个典型的例子来具体描述一下社会学中的小世界现象。

大量的实验表明，一条成功链的平均中间步数是在 5 步到 6 步之间，因而它也被称为“6 度分离”现象。在通信网络中，如果网络中的任意两个结点都能以短跳距相连，即可以通过少量的链路往返在网络中定位存储于任何随机结点中的信息，这个网络就被认为展示了小世界现象。小世界现象是大量网络在自然和技术上所呈现的一个特征，也包括现今的 P2P 网络[16]。

模拟小世界现象的小世界模型经过了三次发展历程。最早的一次是 Pool 和 Kochen 模型[17]的建立。在这个模型中每个结点都与少数的结点相连，这些结点一律是从整个结点集合中随机选取的。它是基于随机网络的网络直径较低这一原理。然而，它无法做到使结点真正的邻居成为它们的邻居。为了解决这个问题，Watts 和 Strogatz 提出了一个新的模型[18]，如图 3-8 所示。在 Watts 和 Strogatz 模型中，网络的边缘被划分为两种连接：本地链接和长链。对网络中的每个结点，它们有小部分最近的邻居作为本地短链接同时还有从结点集合中随机选出的长链。这种方法改进了 Pool 和 Kochen 模型，使得网络直径较低并且结点能够维持它的真正本地邻居。但是，Watts 和 Strogatz 模型并不能保证期望的短路径长度。最近，这个问题被 Kleinberg 的工作[12]所解决。

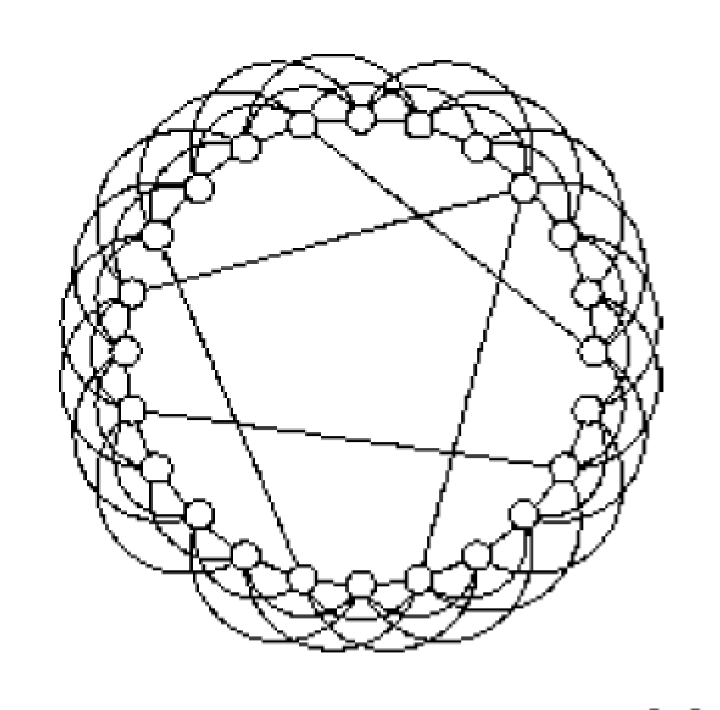
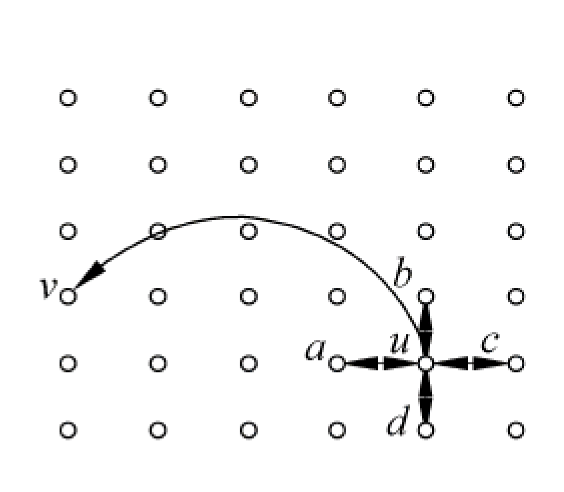


图 3-8　Watts 和 Strogatz 模型[18]

Kleinberg 普遍化了一类小世界网络模型。这个模型提出结点 S 应以 L^{-r} 的概率选取长链 T，其中 $L=\|S-T\|$ ①是 S 和 T 两结点间的曼哈顿距离，r 是底层拓扑的维度。Kleinberg 将长链的分布称为 the Inverse rth-Power Distribution。它给出了甚至在仅有一条长链的情况下，$n\times n$ 个结点的二维网格中路径长度具有上限 $O(\log^2 n)$。它也合理地解释了在 Watts 和 Strogatz 模型中为何无法保证期望的短路径长度：随机选取长链的概率独立于结点在底层拓扑中的位置。Kleinberg 模型如 3-9 所示。



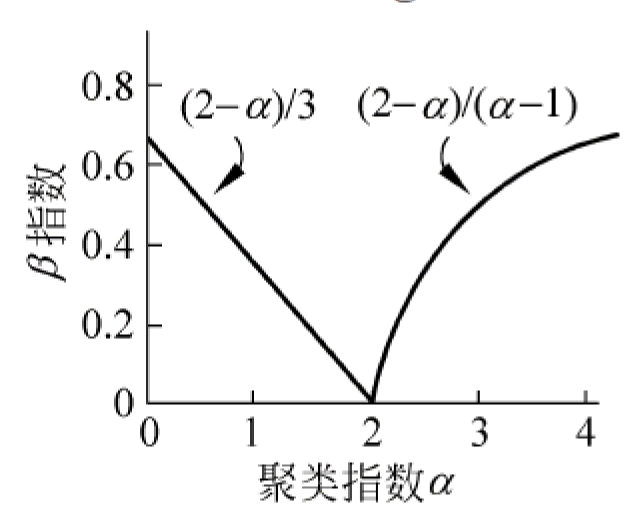


图 3-9　Kleinberg 模型[12]

---

① $\|n_1-n_2\|$ 定义为两结点 n_1 和 n_2 间的曼哈顿距离。对于 d 维网格空间的 X、Y 两点间的曼哈顿距离是 $\sum_{i=1}^{d}|X_i-Y_i|$。

3.2.3　PCCAN

下面将介绍 PCCAN 的拓扑模型及其小世界网络构造。PCCAN 在拓扑上与 CAN 相似，要特别关注的是长链的引入。

1. 拓扑模型

PCCAN 系统模型是在 CAN 基础上提出来的改进模型，所以它的拓扑结构类似于 CAN。首先，PCCAN 在 d 维超环空间配置端结点，d 是正整数表示拓扑结构的维度。PCCAN 使用 DHT 模式且整个 PCCAN 空间被系统结点所划分。每个对象被分配了一个选自于键空间 M 的键值，键空间是通过将 d 维超环空间映射为笛卡儿坐标空间形成的。假设结点空间 N 中有 n 个结点且有 $N \subset M$。每个结点存储对应一定比例键空间对象并且使用一张路由表将某一个不属于结点键空间的对象请求传递到合适的下一跳结点。

PCCAN 的路由表包括两部分：本地短链和缓存的长链。首先只讨论在二维网格中分布的结点缓存一条长链的情况，然后将其扩展到拓扑结构的维度 $d>2$ 及缓存多条长链的情况。PCCAN 使用同 CAN 的贪婪路由算法将查询消息传递到最接近于查询 ID 值的结点。

图 3-10 是一个二维的 PCCAN 系统拓扑结构图，在图中 12 个结点分布在 8×8 的网格中，其中结点 n_4 的路由表显示在图中。这里以表格的形式罗列如表 3-1 所示。

表 3-1　结点 n_4 的路由表

本地短链	缓存长链
n_2、n_6、n_3、n_7	n_{12}

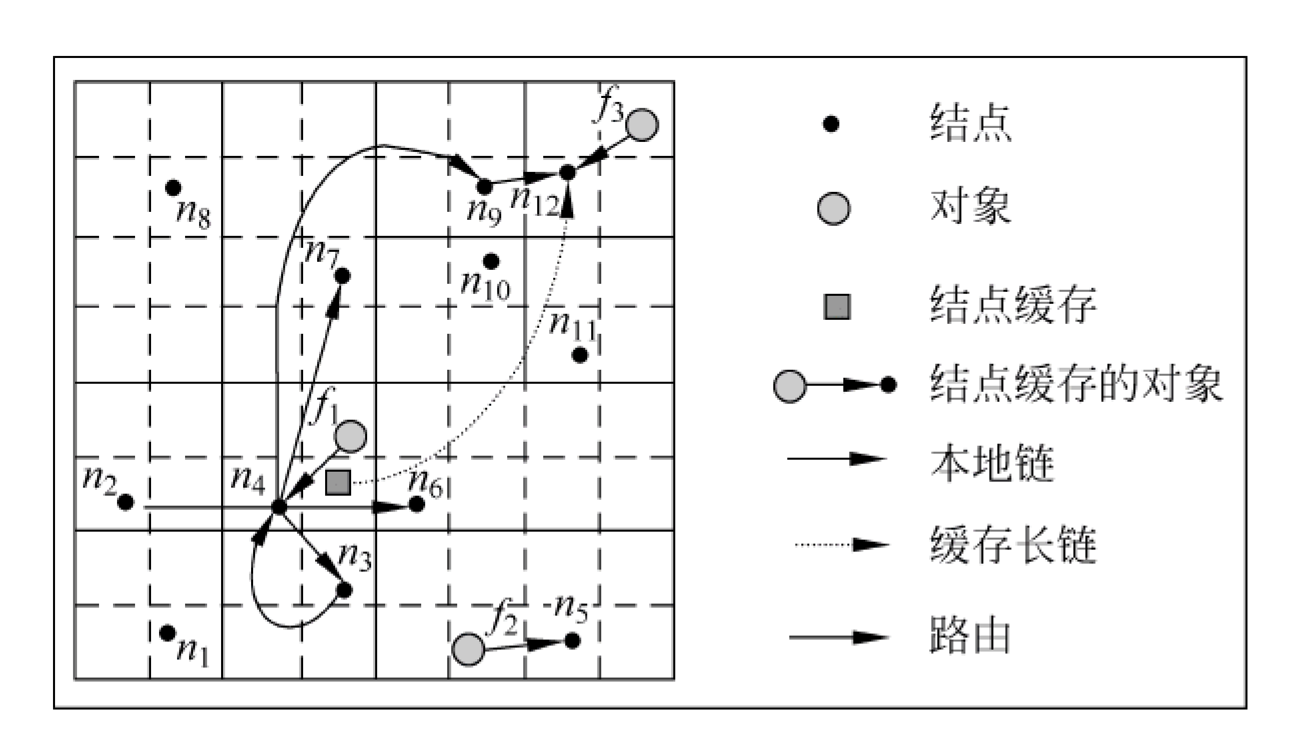


图 3-10　二维 PCCAN 系统拓扑结构

除此之外，每个结点存储的对象也在图 3-10 中表示出来。下面举例说明缓存长链对查询消息路由可能产生的影响。假设结点 n_4 要查询对象 f_3，f_3 被结点 n_{12} 所保存，在没有缓存长链的情况下，n_4 根据自己在路由表中存储的有关邻居区域信息，以及查询消息中目标结点的区域信息，通过距离计算得出与结点 n_{12} 距离最近的邻居结点是结点 n_7。依据贪婪路由算法，结点 n_4 将消息传递给了结点 n_7，结点 n_7 收到消息后再以同样的方式将消息传递给结点 n_9，最终结点 n_9 将消息传递给目标结点 n_{12}，查询完成。在这种情况下，查询消息经过了 3 跳。现在考虑结点 n_4 加入了缓存长链的情况，结点 n_4 可以借助长链快捷到结点 n_{12}，并将消息传递给它，查询在经过 1 跳的情况下完成。对比之下，在结点缓存了长链之后，查询消息经过的路径跳数会在某些情况下减少(当缓存长链与目标结点是同方向时)。但是随机缓存长链不能够使系统路由整体得到优化，因为它不能够提供给系统与其拓扑相

关的路由线索(Routing Clue)[12]。因此,要使得系统路由整体性能得到优化,需要使得系统的缓存长链有小世界的分布特性。

2. 小世界网络构造

PCCAN 系统的小世界网络是利用结点的缓存长链,采用基于蠕虫路由的概率置换策略而构造的。

下面介绍缓存置换过程。系统运行时,某结点 S 应答来自结点 T 的询问请求。假设结点 S 已经将结点 K 作为缓存长链(K 可能是缓存长链的初始值,也可能是缓存长链经过多次置换后的值)。在应答来自于 T 的请求时,它将以 $P=\|S-K\|d/(\|S-K\|d+\|S-T\|d)$ 的概率将 K 置换为 T,否则它将继续保持这个值。这种置换采用的是一种概率置换策略。在路由过程中,转发结点总是用发起查询的结点执行概率置换,即概率公式中的 T 总是查询消息的发起结点。由于概率缓存置换的过程是附加于查询消息的路由过程中的,也就是说凡是查询消息途径的结点都要发生这种概率缓存置换的过程。这种方式像一个蠕虫式的路由过程,我们称之为蠕虫路由机制中的概率缓存置换策略。

图 3-11 为 PCCAN 采用蠕虫路由的缓存置换机制的示意图。结点 n_1 查询 f_3,缓存置换沿查询消息的转发路由被触发。PCCAN 系统在查询消息的路由过程中以概率 p 置换缓存长链,所以结点 n_4、结点 n_6、结点 n_{11} 和结点 n_{12} 在传递此查询消息的过程中,都会以一定的概率缓存查询消息的发起结点 n_1。相关结点路由表的某种可能的变化(因为受概率的影响)情况如表 3-2 所示。

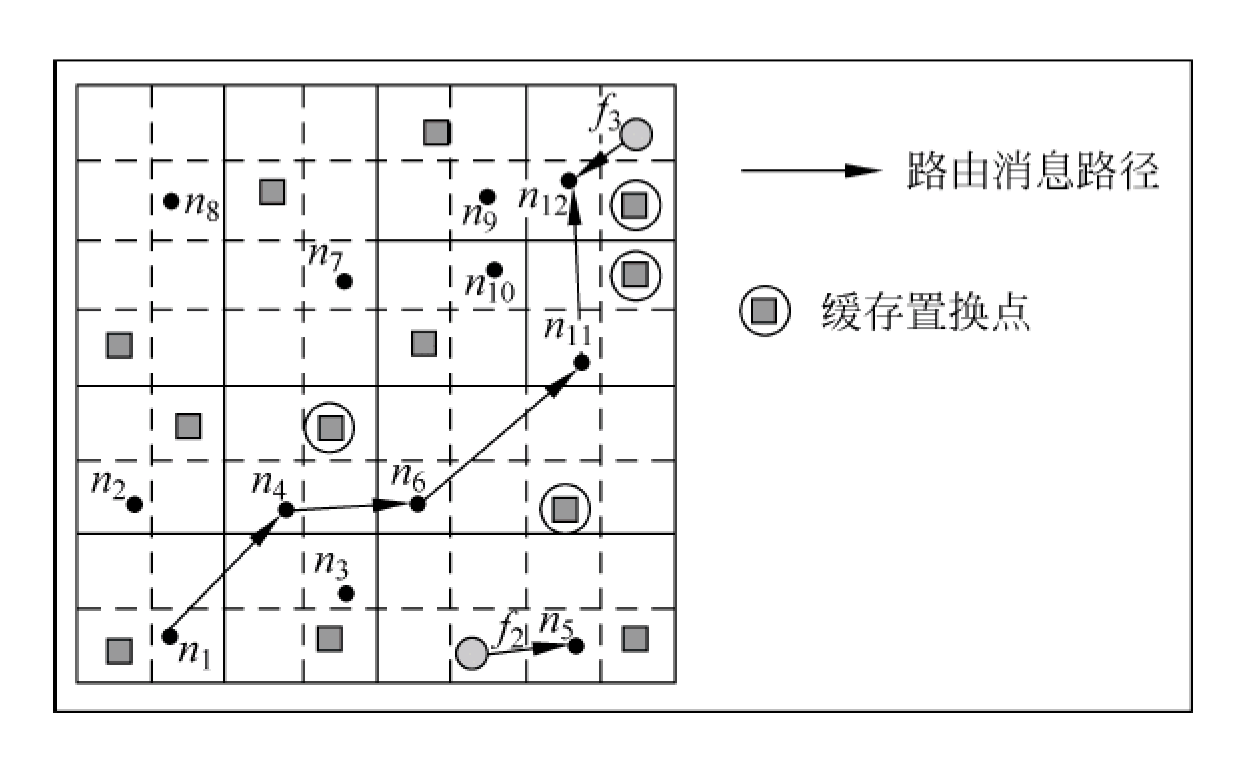


图 3-11 PCCAN 基于蠕虫路由的缓存置换

表 3-2 结点路由表的变化情况

结　点	本地短链	缓存长链初值	置换后缓存长链
n_4	n_2、n_3、n_6、n_7	n_5	n_1
n_6	n_4、n_5、n_{10}	n_5	n_5
n_{11}	n_6、n_{10}、n_{12}	n_8	n_1
n_{12}	n_9、n_{11}	n_7	n_7

在一次查询消息的路由过程中,查询消息的发起结点可能被沿查询路径的多个结点置换进它们的缓存长链中,并且根据概率置换公式,离发起结点越近的结点的置换概率越高。

由于结点缓存长链的置换过程是伴随于查询消息的路由过程，所以静态置换过程附加于结点的额外开销近似为零。此外，采用蠕虫路由机制中的概率置换策略可以让置换沿查询路径进行，从而可以使置换过程更为均匀、迅速，并有效减轻可能的“热键”问题。

直觉上，小世界网络可能由此构成。因为这种概率置换与结点在底层拓扑的位置相关，并且更趋向保留短距离的结点，这正是 Kleinberg 模型的本质体现。下面将通过分析来证明这种直觉的正确性。

3.2.4 分析

以下分析基于的 PCCAN 的模型参数是：结点空间为 N，结点总数为 $n=\pi^d$，其中 d 为 PCCAN 的维度。

1. 静态情况

假定系统内结点总数不变，没有结点加入或者离开，采用概率缓存长链，有如下定理成立。

定理 3-3　PCCAN 系统中重复执行概率缓存置换过程，任意一结点 $s(s\in N)$ 将在有限步内以与 $\|s-t\|^{-d}$ 成比例的概率缓存结点 $t(t\in N\cap t\neq s)$。

证明：假设系统中有 n 个结点。对任意的结点 $s(s\in N)$ 和结点 $t(t\in N\cap t\neq s)$，讨论 s 缓存 t 的概率。

定义一条 Markov 链如下：

将每一种距离设为一个状态，则结点 s 的状态空间为 $S=\{d_1,d_2,\cdots,d_i,\cdots,d_m,d_i$ 是其余结点距 s 的距离$\}$。

假设结点 s 以距离 $x=\|s-t\|$ 缓存结点 t，就说 Markov 链是在状态 x 中。对应 Markov 链中一步转移是指具有距离 y 的新样本（当一条查询路由消息的到达此结点时）被接收，然后链将以 $x^d/(x^d+y^d)$ 的概率转移到状态 y，否则它将停留在原来的状态。这个过程清楚地定义了一条含 m 个状态的 Markov 链。图 3-12 显示结点 s 只有三个状态的 Markov 链示例。三个状态表明一些结点到 s 有三种不同的距离。作为一种简化的说明，图 3-12 所示的例子只显示了三个结点 t_1、t_2、t_3，它们到 s 正好有三种不同的距离。

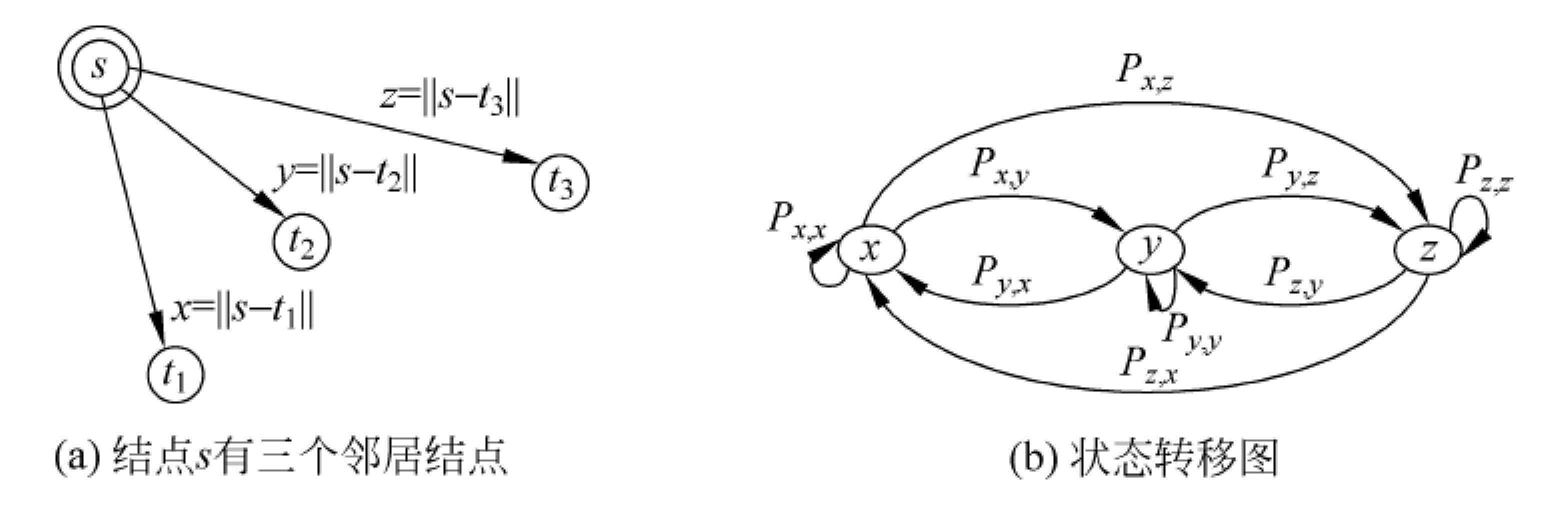


图 3-12　对于结点 s 的仅有三个状态的简化的 Markov 链模型

由马氏链定理可知，对于任何确定数量的状态 m，并且其转移矩阵元素为正数，存在一个稳定且唯一的概率分布（稳态概率）[19]。

设任意一个状态 x 的稳态概率是 p_x。考虑稳定情况下状态 x 的转入和转出过程，注意到 p_x 是一个稳定的值，这是非常重要的，只有这样状态的转入和转出才能达到平衡。这种

平衡说明从状态 x 到状态 $i(i\in S, i\neq x)$ 的所有转出概率等于从状态 $j(j\in S, j\neq x)$ 到状态 x 的所有转入概率。图 3-13 显示了状态的转入和转出的平衡。

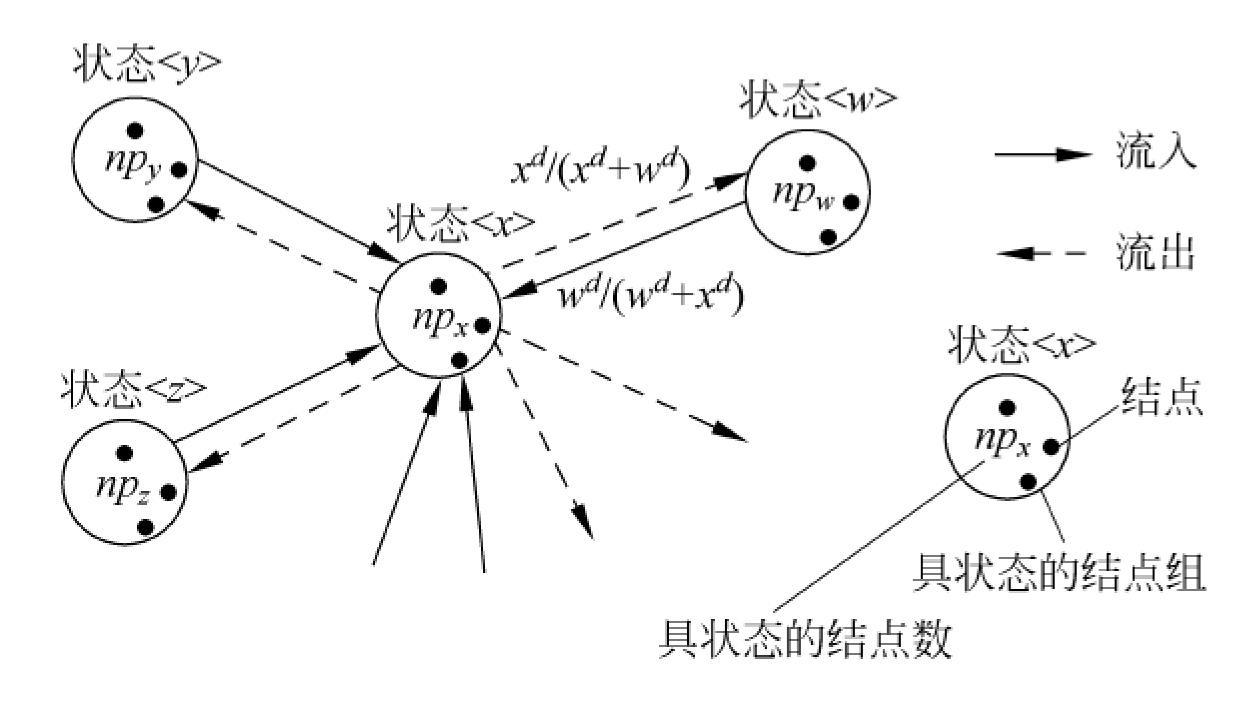


图 3-13　转入与转出的状态平衡图

根据图 3-13，列出方程（方程左边表示转出，方程右边表示转入）：

$$np_x \sum_{i\in S} \frac{x^d}{x^d+i^d} = \sum_{j\in S} np_j \frac{j^d}{j^d+x^d} \tag{3-22}$$

将此式两边消去 n 并通过简单变换可以得

$$\sum_{i\in S} p_x \frac{x^d}{x^d+i^d} = \sum_{i\in S} p_i \frac{i^d}{x^d+i^d} \tag{3-23}$$

式(3-23)是一个对称式，考虑到 $\sum\limits_{i\in S} p_i=1$，当对于任意 $i\in S$ 均有 $p_i i^d \equiv \delta$（δ 是常量数值且大于 0）时，得到一个解：

$$p_i=(1/c)\times\frac{1}{i^d},\quad i\in S \text{ 且 } c=\sum_{i\in S}\frac{1}{i^d} \tag{3-24}$$

现在来证明此解是唯一解。

由于在式(3-23)中，对于一个特定 x 状态，$\frac{x^d}{x^d+i^d}$，$\frac{i^d}{x^d+i^d}$ 对于任意 $i\in S$ 是常数定值，从而式(3-23)是有关 p_i 的常系数方程。设 S 有 m 个状态，则对于任意一个特定状态 $x\in S$，式(3-23)成立，从而有 m 个 m 元方程组，显然这样的方程组仅存在唯一解。事实上，这与 p_i 作为马氏链的稳态概率唯一性是一致的。

式(3-24)表明了概率 p_i 与距离成反比，概率分布服从 Kleinberg 提出的 the Inverse r^{th}-Power Distribution，这正是我们想要的。对所有的状态 $x(x\in S)$ 有稳定的概率 p_x，而状态 x 表示结点 s 和结点 t 间的距离，有 $x=\| s-t|$，因此，对任意的结点 s 属于 N，经过多次缓存置换过程，结点 s 将以与 $\| s-t\| -d$ 成比例的概率缓存结点 $t(t\in N\cap t\neq s)$。证毕。

这样得到的缓存长链是符合 Kleinberg 小世界模型分布的，将在结构化的路由表上构造小世界网络，以缩短查询路径长度。因为稳态概率是唯一的，系统总是收敛到想要的概率分布。并且由于马氏过程与初始值无关，从而无论初始缓存的长链是什么，在有限步内，系统拓扑总是向小世界网络收敛。也就是说，一旦执行了足够的随机查询，系统拓扑就将形成想要的小世界网络。

2. 动态情况

与上述不同，下面考虑的是动态情况，即与现实更吻合的情况。在一个现实系统中，结

点以泊松率 r 加入和离开，总结点数相对稳定。定理 3-3 说明了静态下的小世界收敛，在动态的情况下，仍然期望这种收敛。但是，由于动态下结点的动荡性，一种严谨的定理证明形式将引入多参量和复杂化，这并不是我们当前想要做的工作。这里，我们仅给出定性的分析，并通过实验来检测。分析如下：

(1) 首先，马氏过程本是一个动态过程，因此即便是结点的动态加入与离开，如果能够有足够的相对静态时间，系统仍然能够如同静态情况下，收敛于小世界网络。

(2) 由于收敛受到结点运动加入与离开影响，在最差时，结点动态频率很高(不停加入离开时)，查询置换基本失效；但在最好时，结点加入离开时间上断续，有足够时间让查询置换作用，从而仍然在相对稳定的状态下使网络的缓存链分布动态收敛于小世界模型分布。因此，查询置换率和结点的泊松率(平均加入离开率)是关键。

(3) 假定系统中结点的平均会话时间为 s(单位为 min)，则结点泊松率 $r=1/s$(单位为 min^{-1})，并设它的平均查询时间为 q(单位为 min)，则它的查询置换率为 $t=1/q$(单位为 min^{-1})。也就是当 $q=s$ 时，在整个结点存活的会话时间 s 是一次置换。倘若静态情况下马氏链稳定所需要的步数即需要查询置换次数为 m，则动态收敛应期望在结点的会话时间 s 内能有 m 次置换发生，因而有：

$$s = q \times m \Rightarrow t = r \times m \tag{3-25}$$

式(3-25)说明查询置换率与结点的泊松率成正比。因此，要减轻结点动态性对于收敛的影响，需要尽可能提高查询置换率。

(4) 尽管提高查询置换率能够减轻结点动态性的影响，但是相应也增长了附加开销。增长的附加开销包括处理开销及网络带宽开销。由于当前计算机趋向是存储的不断增长以及 CPU 处理能力的提高，因此，附加的开销主要在网络带宽上。可以假设一条查询置换消息占用带宽为 b(单位为字节，即 B)，则为了近似静态收敛效果，设 1min 内的查询置换次数为 t，满足式(3-25)，从而查询置换消息占用带宽是 $t\times b$(B/min)，并由于所用的时间是 s(结点会话时间)，附加总带宽为 $t\times b\times s$(单位为 B)。设结点当前线路带宽为 w(单位为 B/min)，从而结点平均占用线路带宽率为

$$f = \frac{tb}{w} = tbw^{-1} = rbmw^{-1} \tag{3-26}$$

因此，结点平均占用线路带宽率是与结点的平均查询置换率成正比。

综上所述，在动态情况下的收敛依然是可行的，但是必须相应增长查询置换率，其增长的查询置换率与结点的泊松率成正比；增长查询置换率增大了网络带宽开销，其平均占用线路带宽率同样正比于查询置换率。需要在现实情况下进行折中考虑。

3. 路径长度

定理 3-4　*采用概率缓存模式的小世界网络构造，PCCAN 的路由平均路径跳数为 $O(\log^2\pi)$ 即 $O(\log^2(n^{1/d}))$，其中 n 是结点总数。*

证明：PCCAN 结构类似于 Kleinberg 提出的小世界结构。它们都有相似的本地短链接和一条长链。尽管 PCCAN 的长链是在缓存中，定理 3-3 证明了采用概率缓存模式，缓存的长链将以一种可行的途径服从于反 r 次幂律分布(the Inverse r^{th}-Power Distribution)，也就是说，对于一结点 u 选择结点 v 作为它的长链的概率 p 为

$$p = d(u,v)^{-d} / \sum_{v \neq u} d(u,v)^{-d} \tag{3-27}$$

因此,参照参考文献[12]中定理2的一个相似的证明过程,可以得到路由的平均路径跳数是 $O(\log^2 \pi)$,即 $O(\log^2(n^{1/d}))$。证毕。

相比较CAN路由性能,PCCAN在保持 $O(1)$ 路由状态下,将平均路径跳数由 $O(n^{1/d})$ 改进到 $O(\log^2(n^{1/d}))$。

3.2.5 讨论

1. 动态适应

我们在3.2.4节对于动态收敛情况作了定性分析,并认为提高查询置换率是关键所在。实际上式(3-26)给出了在结点动态情况下期望的查询置换率的下限,当超出此限制时,结点在动态下不仅收敛,而且速度加快。

根据3.2.4节的分析,应有:

$$s \geqslant q \times m \Rightarrow t \geqslant r \times m \tag{3-28}$$

$$f = \frac{tb}{w} = tbw^{-1} \geqslant rbmw^{-1} \tag{3-29}$$

我们期望的是查询置换率能够满足式(3-28)并且有较小的 f。

当现实系统布置时,查询置换率是由系统里的查询请求情况所决定的。因而,当查询请求较少而网络动态较强时,很可能不能满足式(3-28)的条件。因此,为了使得动态收敛可行并且可加快收敛过程,我们设计了主动查询机制的技术。主动查询机制根据式(3-28)与当前查询置换的实际情况来主动地增加查询请求以适应动态收敛的要求。主动查询算法如下(静态稳定步数 m、查询置换率下限 ζ、主动加速率 Δ 及算法调用默认时间间隔 θ 为系统配置的已知参数值):

(1) 测量当前的查询置换率 $\bar{t}$。结点根据当前自己的单位时间内查询置换次数平均值来估计 $\bar{t}$。

(2) 测量当前的泊松率 $\bar{r}$。结点根据单位时间自己的邻居结点的平均失效率可以估计出泊松率 $\bar{r}$。

(3) 根据式(3-28)判断是否满足 $\bar{t} \geqslant \bar{r} \times m$。如果不满足,计算需要的主动查询率 $\varepsilon = \bar{r}m - \bar{t} + \Delta$;如果满足,则还要考虑是否满足 $\bar{t} < \zeta$,如果不满足,计算需要的主动查询率 $\varepsilon = \zeta - \bar{t} + \Delta$,而如果满足,则算法结束,算法调用的下一时间间隔为默认值 θ。

(4) 当前结点以 ε 的主动查询率向网络中随机选择的结点主动发送特定查询请求,并设置 $(1/\bar{r})$ 为下一次调用算法的时间间隔。此特定查询请求的特点是:不要求返回和请求资源,它仅标识源发起点属于活动状态。因而,和普通查询相比,减轻了开销。

主动查询算法的(4)中通过让算法调用时间间隔与算法本身相关,从而避开了如同Chord的Stabilization算法在动态情况下不能自适应的问题。并且,当系统查询充分时,并不激活主动查询的发送,从而尽可能减少了网络开销。事实上,实际系统中查询请求频率往往比结点的加入退出的频率高出好多数量级,因而发送主动查询并不是经常必须的事情。

此外,还要考虑主动查询机制的现实性,它引起的网络开销是否过大,可行度如何?为了说明问题,我们举个例子。参考文献[2]中通过Overnet[20]测量有50%结点的会话时间小于1h,但也说明有50%结点会话可能超过1h。为了说明现实的可行性,不妨让条件苛刻

些，让会话时间在 1min 到 60min 范围，并且假定结点的线路带宽为低端 Modem 下的 56 000b/s 下，看 f 是否在可接受的范围内。一条主动查询的特定消息不妨假定为 100B，m 可以假定为 50(如实验所测)，查询率不妨放大设为 2 倍于泊松率，这样下来，f 的范围是 [0.0004，0.023]。这说明即便很严苛的情况下，主动查询开销所占用网络带宽资源也是很少，其是非关键性的。

2. 均衡置换与收敛速度

查询置换利用蠕虫路由的实质是放大一条查询的效用，使得收敛速度加快。此外，它也能有效减轻“热键”行为，使置换在系统内均匀。因为虽然有流行的对象可能会接受很多请求，成为“热键”，但对哪个键进行初始请求的结点可以被认为是均一地从网络中选取的。使用蠕虫路由置换机制，每个结点的缓存置换发生概率将近似趋于相同。

小世界网络收敛受查询请求率影响，而设计的主动查询机制则有很好的自适应，以保证收敛速度。当网络查询请求少时，低于下限 ζ，将激活主动机制，增加主动查询率；而网络动态时，则自适应调节主动查询率。此外，还提供了主动加速率 Δ，以便更好地利用应用的特点。静态稳定步数 m、查询置换率下限 ζ 及主动加速率 Δ 均为系统参数，系统可以根据实际情况进行调整以更好服务实际应用。

3. 路由表维护与概率效应

PCCAN 的路由表维护考虑本地短链接和缓存长链的维护。对本地短链接的维护和 CAN 相同，读者可以查阅参考文献[21]得到更详细的说明。对缓存长链，为提高有效利用率可设定两种软状态：有效和无效，并周期性进行软状态更新。

缓存长链的及时更新对于系统路由性能是重要的。除了正常的周期更新外，还可以利用概率效应。所谓概率效应是指由于结点利用了基于蠕虫路由的缓存置换策略，从而在每次查询到达时都会利用当前缓存长链进行概率更新，导致失效长链将以一定概率被当前的活动查询更新，而更新后的缓存长链是活动的。概率效应可以大大提高缓存长链的有效性，特别在主动查询机制的作用下，更能够适应动态网络环境下的路由表维护。

4. 多条长链

上面只考虑了一条缓存长链的情况，现在扩展到 $k(k\geqslant 1)$ 条缓存长链的情况。这时，每一结点都缓存 k 条长链。并当一条路由消息到达时，它将当前结点所有缓存的 k 条长链同时用到达的查询消息进行置换，以保持收敛速度与缓存仅一条长链时的情况相似。同样的，它也是用查询初始结点来与缓存的长链进行概率置换的。k 条缓存的长链可以看作是定理 3-3 所描述的 Markov 链中的 k 个状态，由于它们都参与了马氏过程，因而比单链具有更快的收敛性。

随着长链数量的增加，对一个查询的路由可以更多的短路径选择，因而平均路径跳数得到减少。

5. 缓存置换策略

下面考虑不同的缓存置换策略对于路径长度的影响。

对于传统的确定性缓存置换，比较常用的有“最近最少使用算法”(LRU 算法)，它是将最近一段时间内最少被用到的缓存项淘汰。当缓存数据项时，可能会提高数据项的命中率。但是在 PCCAN 中，缓存的是结点用于定位，这样，采用 LRU 算法是无向性更新缓存，将不

可能导致整体性能的优化。

对于本书的概率性缓存置换，也可以试采用 Watts & Strogatz 小世界模型提出的方法。Watts & Strogatz 的模型中，结点 u 选择 v 作为长链的概率是 $1/\sum_{v\neq u} d(u,v)^{-d}$，长链呈一致性分布(Uniform Distribution)。这种长链选择的问题是没有考虑所选结点在拓扑中的位置，从而不能够形成小世界的聚集现象，因而不能够收敛于小世界网络。在参考文献[12]中，给出了它的路径下限是 $\Omega(n^{\frac{2}{3d}})$。

6. 维度影响

考虑 PCCAN 的维度对路径长度的影响，这时我们关心缓存长链的作用。随着维度增加，网络直径将会减小，这样，长链对网络的影响就降低了。因此，PCCAN 更适合布置于低维中。特别的是，当把它弱化为一维时，它的实际效果类似 Symphony[13]。不同的是，Symphony 采用主动长链构造，而 PCCAN 采用的是缓存构造。

3.2.6 实验

本节将通过 PCCAN 仿真实验得出相应结论。本处所用仿真器是用 Java 编写，并在 Linux 平台下测试的。

分别测试 PCCAN 在静态情况和动态情况下的收敛过程及路由性能。结点将它加入网络时的联系结点作为初始的长链，当有一条查询消息到达，它在传递消息的同时利用概率置换策略更新结点的缓存。仿真是以二维 PCCAN 为基础，具有与 Kleinberg 相近的拓扑结构。PCCAN 运行在 64～4096 个结点的网络且这些结点在网络中是随机分布的。

1. 静态情形

在静态实验中，理想的情形是网络中不存在任何结点的加入或者离开。当 N（N 从 64 到 4096 变化）个结点加入网络后，从中随机选取两个结点，其中一个向另一个发送查询请求。路径跳数和缓存置换的次数在每一次查询中被记录，当 100 次查询完成后，计算平均路径跳数和每个结点的平均置换次数。

首先讨论二维静态 PCCAN 网络的收敛情况，其中网络的初始结点数设为 $N=1024$。并比较结点缓存不同长链数、运用不同的缓存置换策略及 PCCAN 维度对于路径长度路由性能的影响。

1）单条长链

考虑理想的情形即网络中不存在任何结点的加入或者离开，当 1024 个结点加入网络后，从中随机选取两个结点，其中一个向另一个发送查询请求。路径跳数和缓存置换的次数在每一次查询中被记录。100 次查询完成后，计算平均路径跳数和每个测量的平均值。从图 3-14 中可以清楚地看出，当全部结点的平均置换次数为 50 次时系统完全达到稳定。这是一个令人满意的结果，它说明每个结点需要至少 50 步使其缓存置换的 Markov 链达到稳定，从而形成小世界网络，这是与定理 3-3 相符合的。

此外，在单链下，还变换了网络中参与结点数目来比较 PCCAN 与 CAN 的路由性能。需要强调的是平均路径长度的改进，变换网络参与结点数从 64 到 4096，并且路径长度是系统收敛后的稳态测试的，结果如图 3-15 所示。观察图 3-15 可知，PCCAN 的路径跳数比

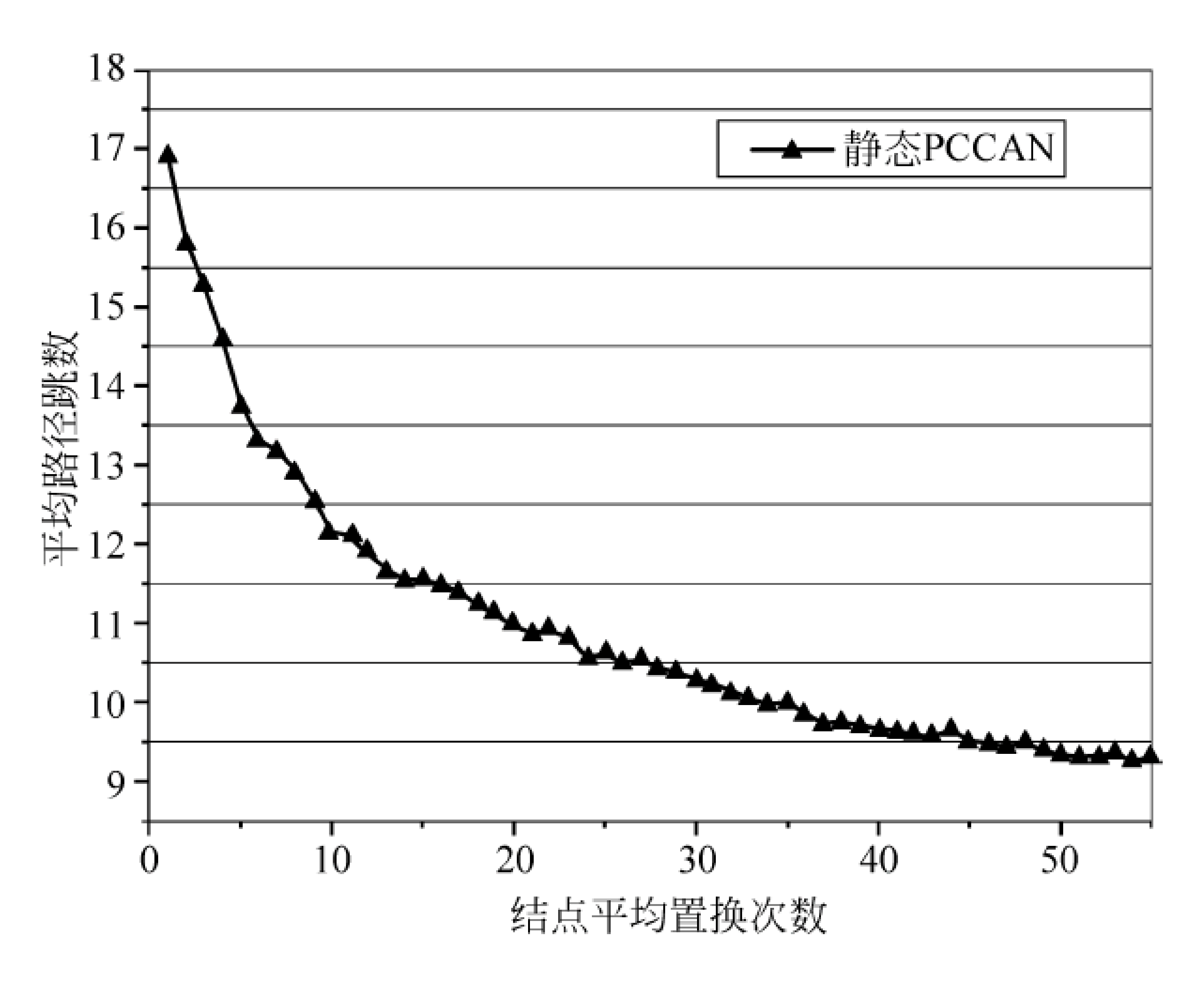


图 3-14　PCCAN 的静态收敛

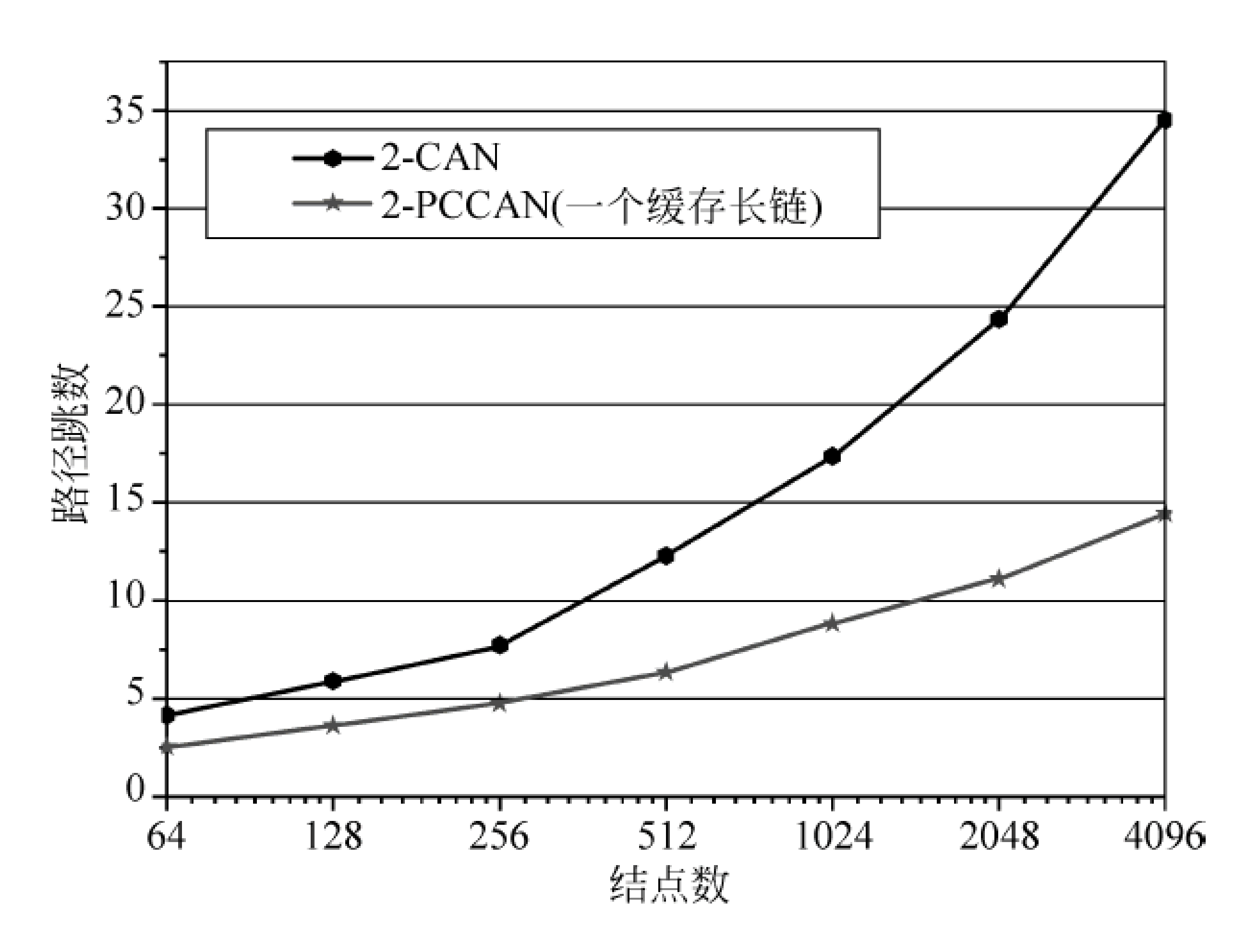


图 3-15　单链下的路径长度

CAN 显著地减少,这与定理 3-4 是一致的。

2）多条长链

我们也测试了多条缓存长链的情况。相对于缓存一条长链的情况,当系统中的每个结点概率缓存置换多条长链时,系统查询消息的平均路径跳数下降趋势更为明显,并且在结点的平均置换次数与单条长链情况相似时,系统查询消息的平均路径跳数也平缓地达到一种稳定的状态(即平均路径跳数在某一个很小的区间内波动)。这说明缓存多条长链可达到小网络收敛并且可以更为有效地减少 PCCAN 系统查询消息的平均路径跳数。在二维 PCCAN 中,网络结点数为 1024 个,变换结点缓存的长链数目,得到实验结果如图 3-16 和图 3-17 所示。其中图 3-16 是将长链数目从 3 变换到 6,并观察其稳定效果;而图 3-17 则是在稳定收敛下比较不同长链数目的路径长度。测量点取 100 次的平均值。

从图 3-16 中可以看到,当概率缓存置换的长链数递增时,系统有更快的收敛性,这是由于多链能够在一次置换时有更多的状态参与马氏链转移,从而加速了收敛。此外,随着长链

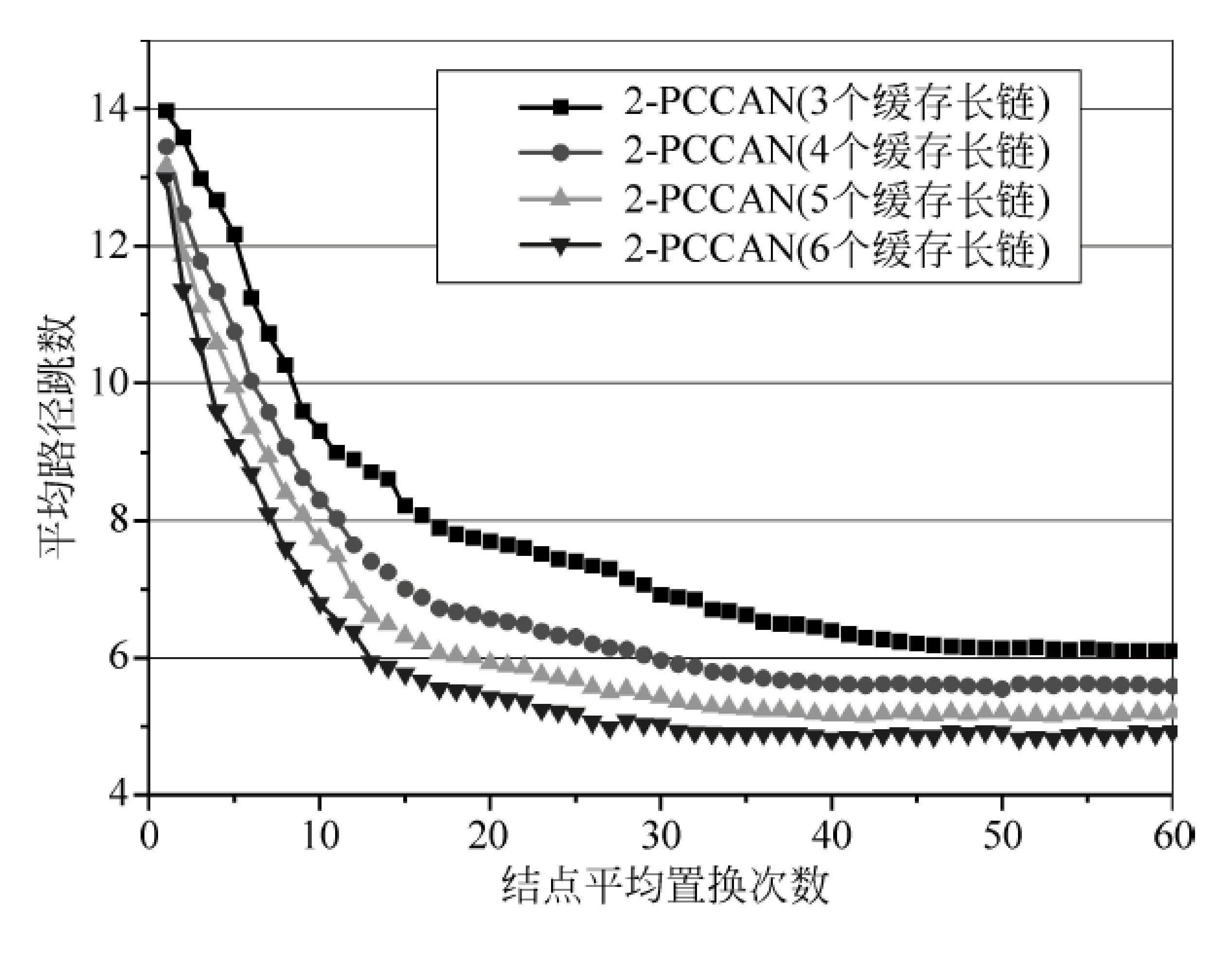


图 3-16　不同长链数目的比较

数目增多，系统查询消息的平均路径跳数在达到稳定状态后的值相对递减，即长链数越多，平均路径跳数越少。这是由于概率缓存的长链数越多，结点在传递查询消息时的路由选择就越多，也就越可能选择到最佳的下一跳结点，从而减少查询消息的路径跳数。但由图也可以观察到递减幅度在不断减少，说明长链的影响也是有限的。随着长链数目增多，改进将越来越趋于平缓，如图 3-17 所示，说明长链对于缩短网络直径的效用达到了一个限值。

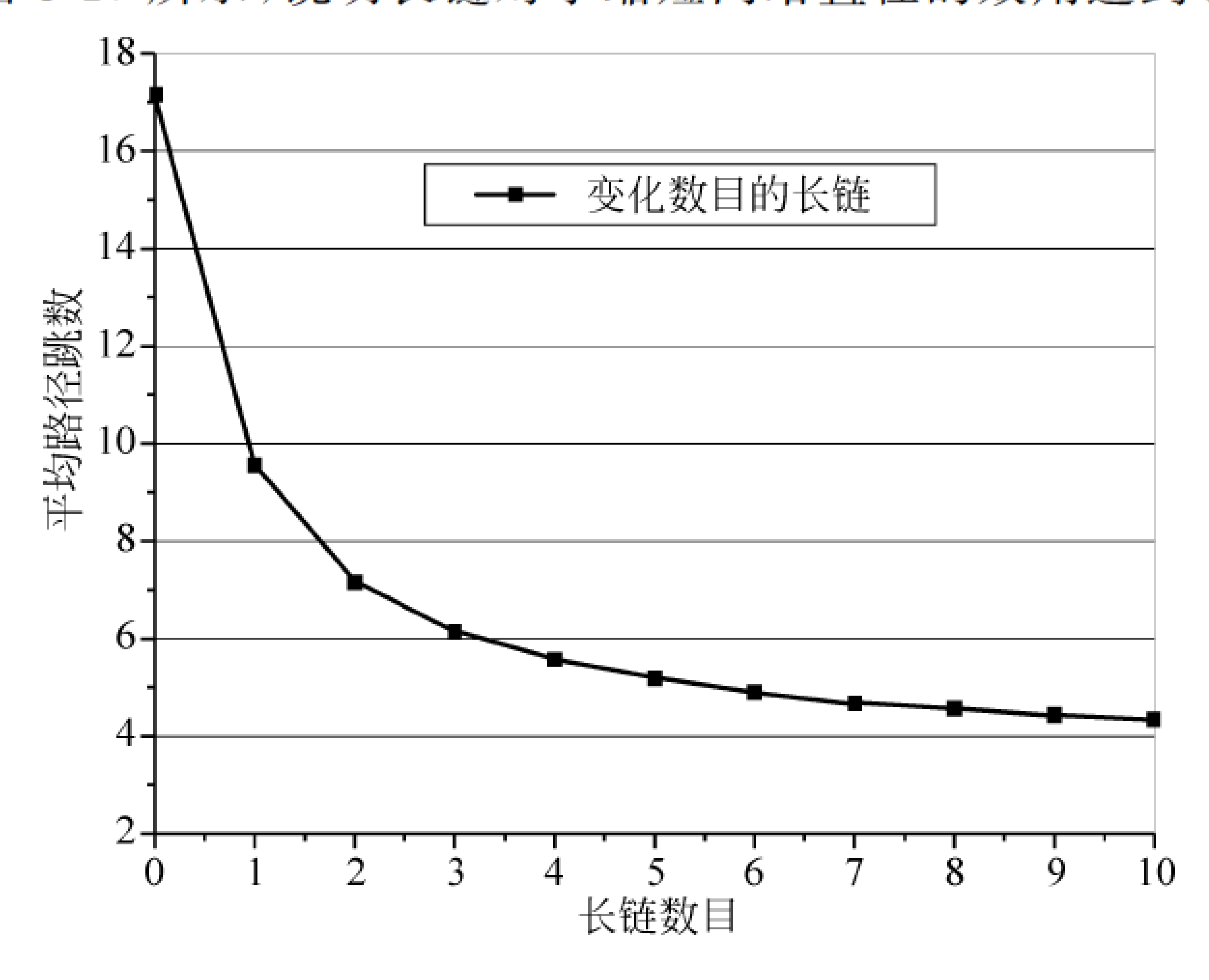


图 3-17　不同长链数的路径长度

3）缓存置换策略

为了更好地说明概率缓存置换策略对于改进网络路由性能的有效性，在实验中对比了运用不同置换策略对系统收敛的影响。PCCAN 中比较了三种缓存置换策略：采用 Kleinberg 模式，采用 Watts & Strogatz 模式，采用 LRU 模式。

在二维 PCCAN 中，网络结点数为 1024 个，结点缓存的长链数目为 3，变换三种不同的置换模式。图 3-18 显示了运用不同置换策略后系统查询消息平均路径跳数统计情况。从图中可以看出，在系统处于相同状态下运用这些不同的置换策略，系统的收敛情况是不同

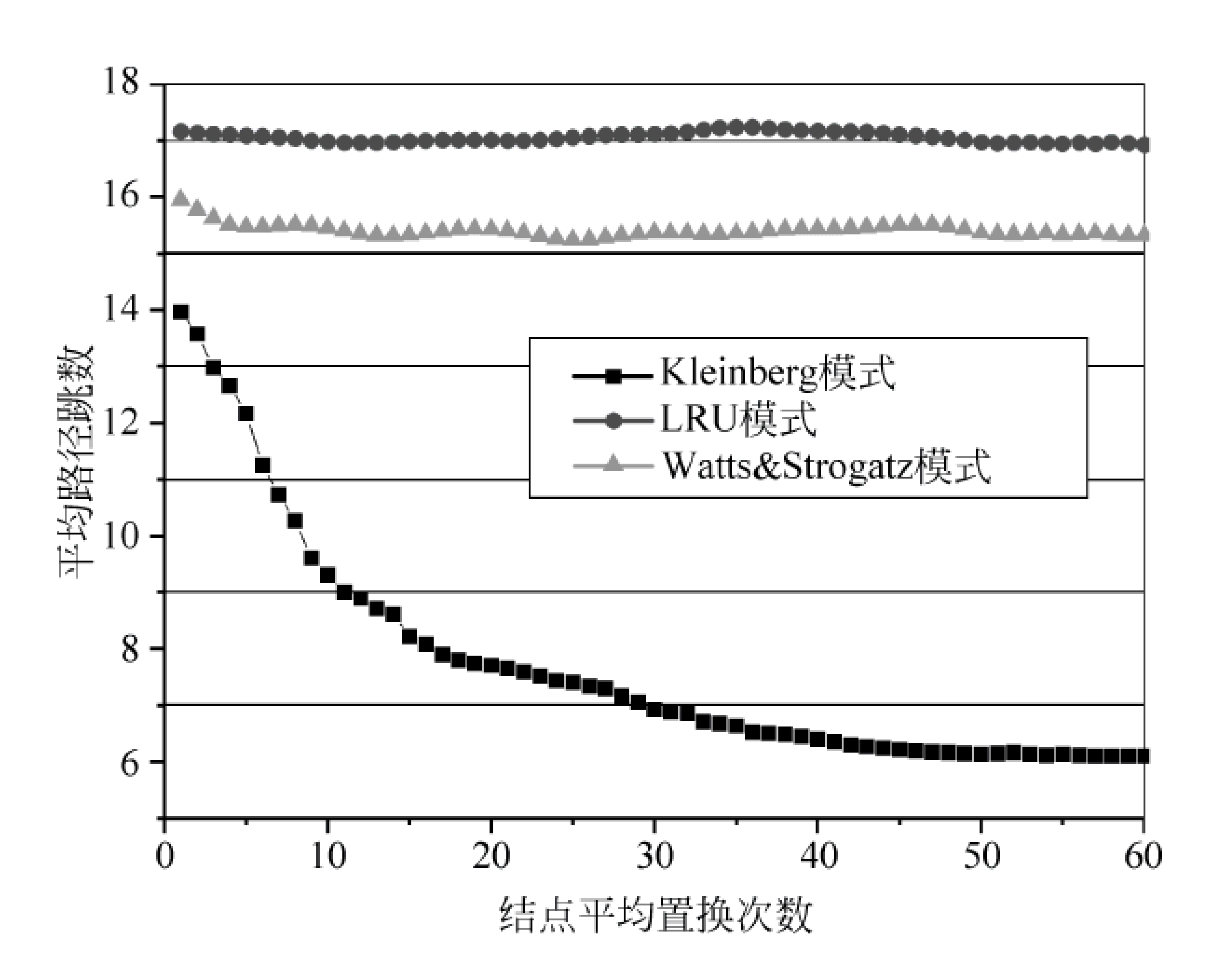


图 3-18 三种缓存置换策略的比较

的。在运用 LRU 模式时，系统查询消息的平均路径跳数不再呈现下降趋势，只是在初始值附近区间内波动；在运用 Watts & Strogatz 模式时，系统查询消息的平均路径跳数有下降趋势；在运用 Kleinberg 模式时，系统查询消息的平均路径跳数呈显著下降趋势，并且在完成了一定次数的查询后，系统逐步趋于稳定的状态。

这些观察结果是与在 3.2.5 节的分析是一致的。LRU 模式是无向性更新缓存，因而对于路径长度的改进几乎不起作用。而 Watts & Strogatz 模式由于缓存长链的建立没有考虑结点的拓扑位置的概率相关性，从而不能够起到“路由线索”的作用，因而改进性能有限。只有 Kleinberg 模式能够较好地捕获网络结点的布置特征，从而将提供路由的快捷方式，较好地减少了路由的路径长度。因而，PCCAN 采用 Kleinberg 模式作为其缓存置换，是最佳的方案。

4）维度影响

图 3-19 所示对比了不同维度对 PCCAN 的路径长度的影响。实验的网络结点数目为 1024，变化 PCCAN 系统的维度参数从 2 到 10。通过比较可以发现 PCCAN 仿真系统的收敛性在低维的情况下表现得更为明显，比如在二维的情况下，PCCAN 系统收敛后的平均路径跳数相比于同等参数设置下的 CAN 有较大幅度减少。这与 3.2.5 节的分析是一致的。

2. 动态情形

本节设置 PCCAN 系统为 1 条缓存长链，并测试动态环境下系统的收敛性和动态适应性。对于一个实际的网络，结点的加入和离开是动态的。仿真实际网络如下。

当 1024 个结点加入网络后，一些结点离开网络，一些结点加入网络，结点的加入和离开以相同的速度。这种方式下，结点的总数仍然是不变的，这是对实际网络的一种普遍假设。为了形成小世界网络，缓存置换机制应该让一个随机新加入的结点有达到稳定的可能性。使用 3.2.5 节提出的主动查询机制。主动查询能够自适应结点的动态性从而保障和加速收敛速度，并且也能够通过概率效应帮助路由表修复失效的缓存链。以下实验中主动查询算法的系统参数配置如下：

静态稳定步数：$m=50$。

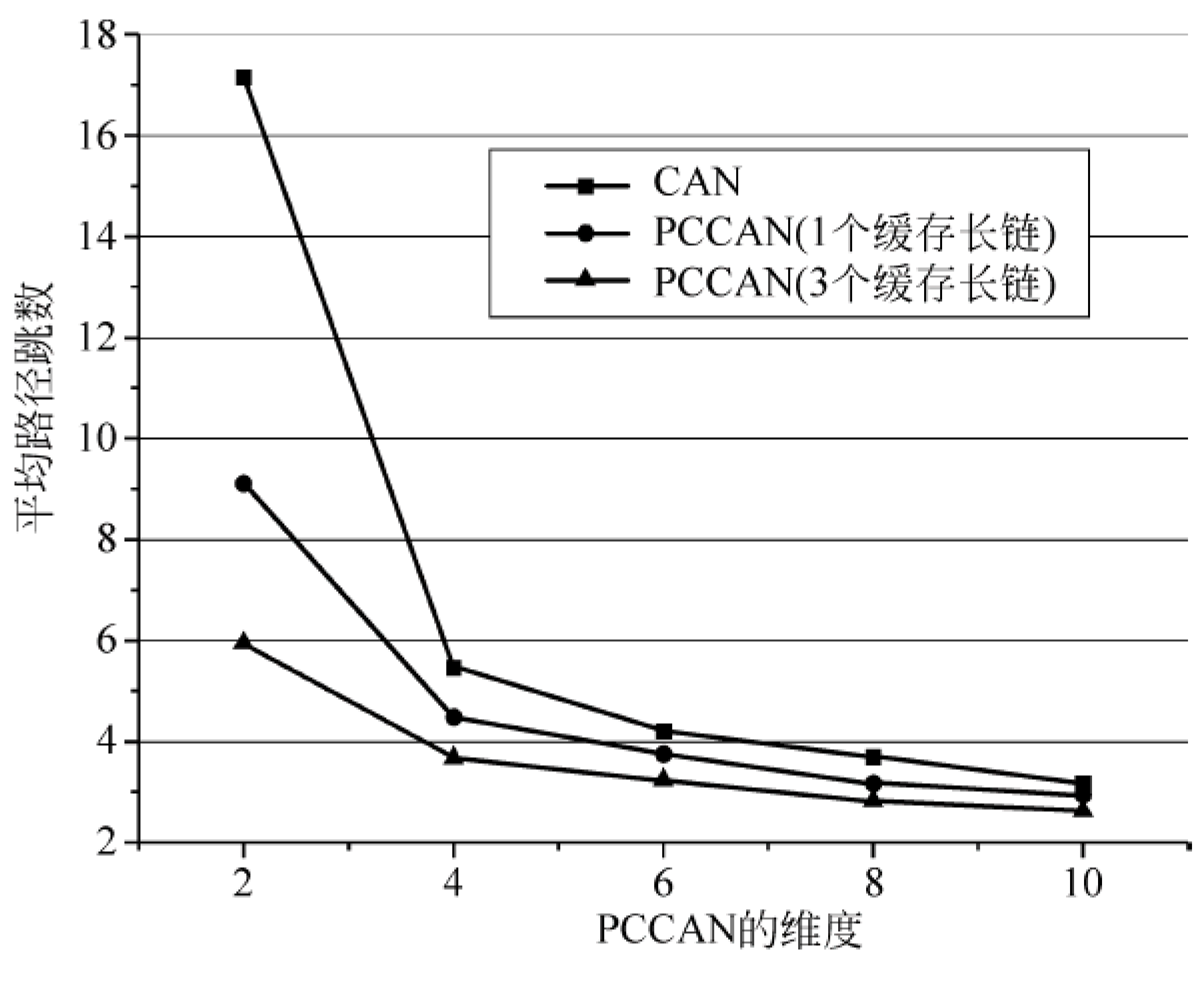


图 3-19　维度影响

查询置换率下限：$\zeta=1$ 次/min。

主动加速率：$\Delta=0.5$ 次/min。

默认调用时间间隔：$\theta=30\text{min}$。

1）动态收敛

让初始的平均结点查询率$\bar{t}=1\text{min}^{-1}$（即结点每 1min 发出一个查询请求），以及结点的泊松率$\bar{r}=0.1\text{min}^{-1}$（即结点加入和离开的速度为 10min 一个）。当主动查询过程调用时，由算法得出由于系统动态性，要补偿的主动查询率 $\varepsilon=\bar{r}m-\bar{t}+\Delta=0.1\times50-1+0.5=4.5\text{min}^{-1}$。这样每个结点在其存活期内（10min）平均可以得到 55 次置换，故在此动态网络中系统仍然能达到较好收敛。

接下来，从网络中随机选取两个结点，其中一个向另一个发送查询请求。路径跳数和缓存置换的次数在每一次查询中被记录。100 次查询完成后，计算平均路径跳数和每个结点的平均置换次数。对于动态模型，计算平均路径跳数时只考虑成功的查询，计算平均置换次数时只考虑当前活着的结点。图 3-20 所示为实验结果。与静态模型相比，会聚过程呈现更明显的波动性，速度也变缓慢了。这对于一个动态环境是一个显然的结果。不过，系统还是以这种动荡的方式到达稳定状态。原因是概率置换策略总是趋向于保留短链接，并使得长链的分布服从反 r 次幂律分布，因此拓扑结构会逐渐朝着小世界网络的方向调整并最终达到一个稳定结构。

2）动态适应

PCCAN 系统利用主动查询机制来适应动态网络环境。实验是通过在变化结点平均会话时间的动态网络下，测量路径长度的波动性及结点的主动查询率，从而检验其适应情况。实验中设置初始结点平均查询率$\bar{t}=1\text{min}^{-1}$，结点的平均会话时间变化为 5min、10min、15min、20min、25min、30min。

如图 3-21 所示，可以观察到平均路径长度随着会话时间变化而在一个范围内波动。这说明尽管会话时间变化导致网络动态，然而主动查询能够自适应网络的动态变化，从而保证了可靠的收敛性，使得路径长度波动性较小。此外，我们还可观察到结点的平均查询率随会

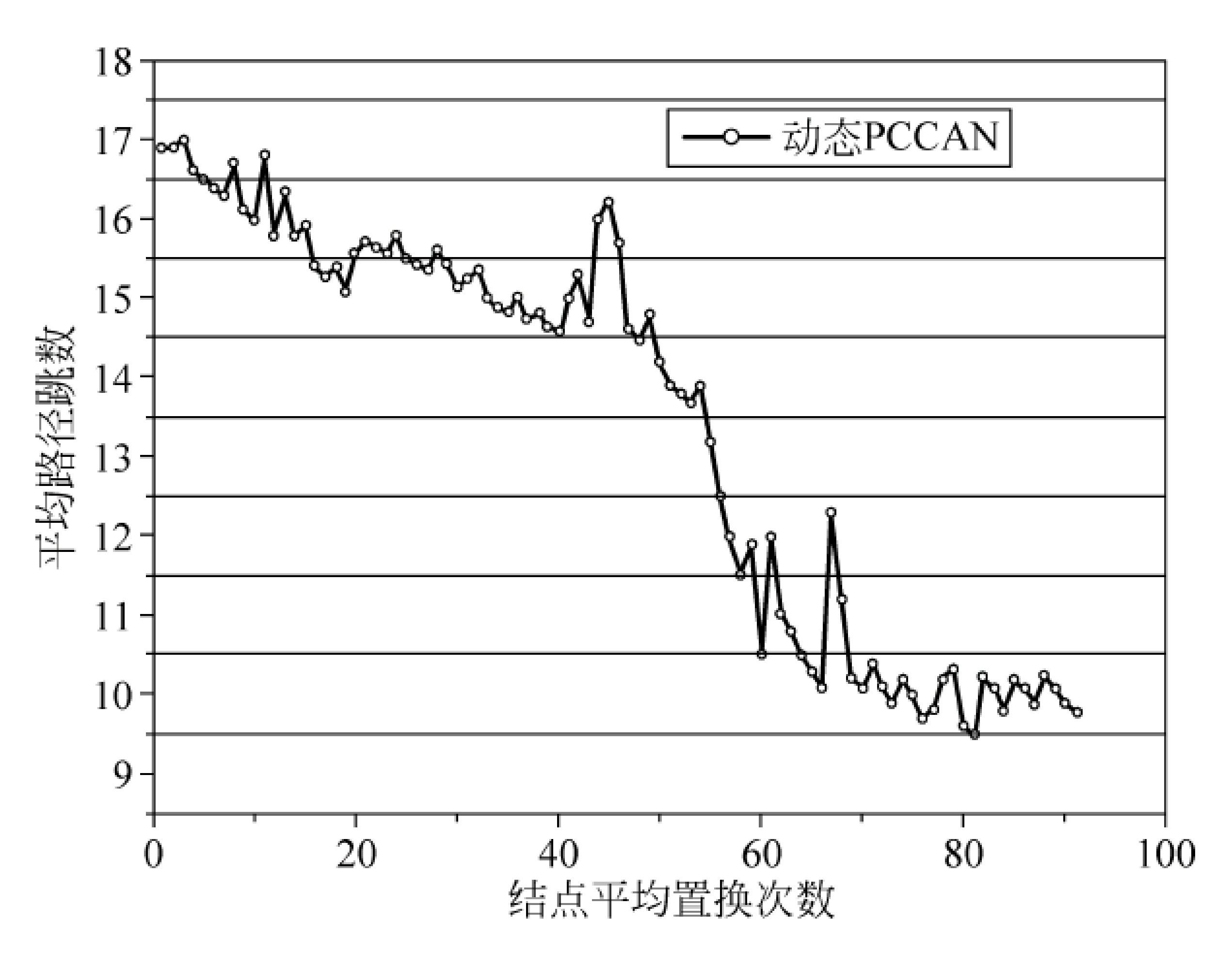


图 3-20　PCCAN 的动态收敛

话时间延长而较大地减少。这是由于结点平均会话时间变大,网络动态性相对减弱,致使主动查询率作为补偿作用也相应大大减少。综合实验的观察,可说明 PCCAN 系统对于动态网络有良好的适应性。

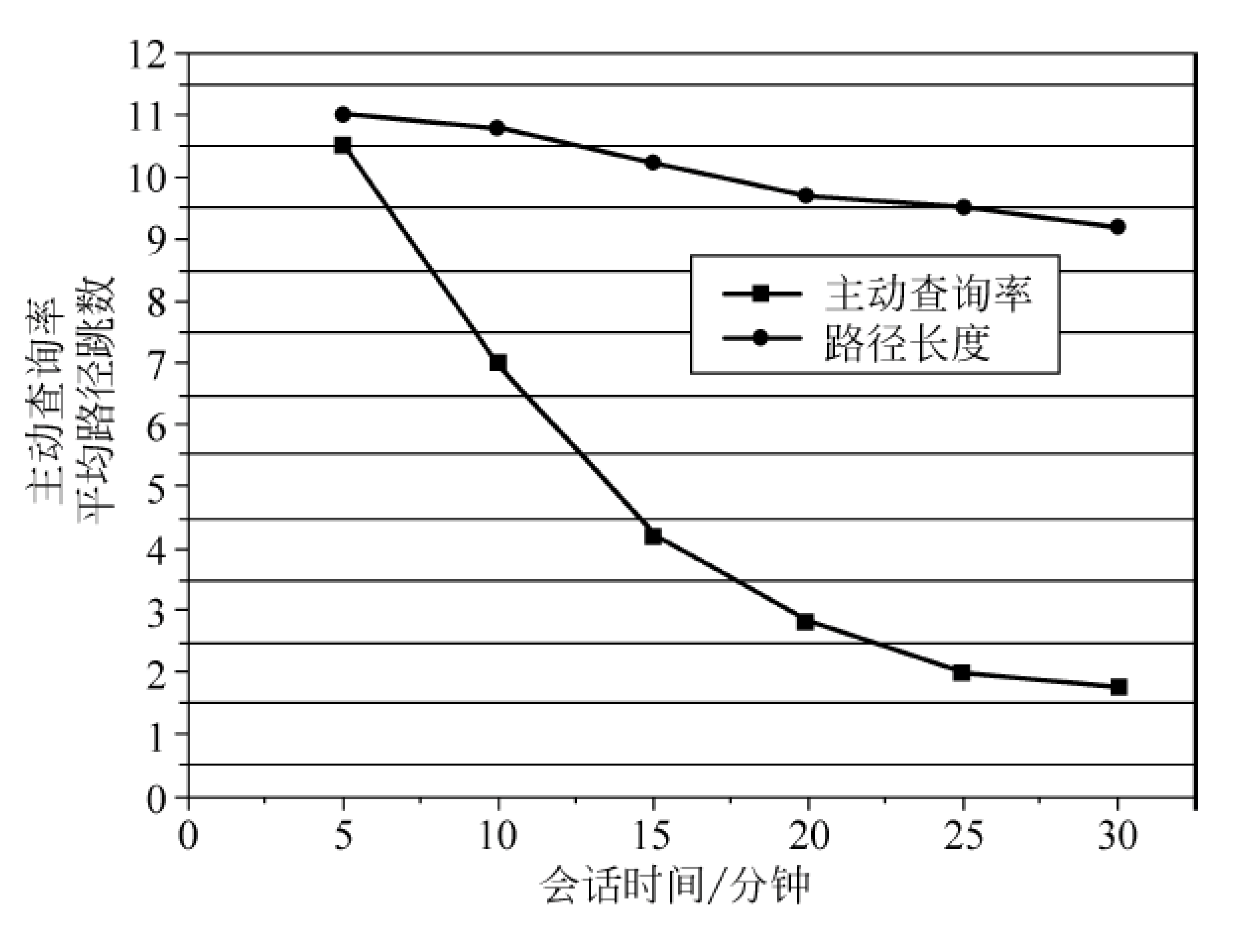


图 3-21　自适应动态网络

3.2.7　相关工作

DHT 路由性能提高的问题得到了相当的重视,特别是其中的状态与效率的问题[5]。小世界网络具有低状态、高效率的特性,因此,将它与路由结合,会是一种上述问题较好的解决之道,这正是本节的出发点。我们基于 Kleinberg 的小世界理论,提出了一种新型的概率缓存长链技术,可以在 DHT 路由表链间构造小世界,以改善 DHT 路由。我们的工作与以往工作不同,具体如下。

Symphony[13]使用了一种 Kleinberg 构造的连续化版本来避免 Kleinberg 构造所需的全局信息。然而,这种连续化模式与我们的概率缓存模式相比,在结点动态加入和离开时需

要相当高的调整开销。Zhang H. 等[21]讨论了一种以概率缓存模式在 Freenet 上形成小世界现象的方法，但是缓存的对象不同。参考文献[21]所涉及的论文中缓存的是资源索引键，它辅助非结构系统中的 Flooding 方式的路由定位；而我们缓存的是结点标识，并成为 DHT 路由表的核心部分，辅助结构化系统的 DHT 路由定位。参考文献[22]提到了一个与我们模式类似的方法。但它所讨论的局限于一维标识空间，并且没有探讨维度的普遍化及动态情形处理等关键性问题。

本节的概率缓存模式是对传统的确定缓存模式的突破。传统的确定缓存模式如 LRU 已经被广泛地应用于非结构化 P2P 系统，以减少网络拥塞和提高命中率[23]。事实上，在应用 Flooding 路由的系统，如 Gnutella、Freenet 中，使用确定缓存模式对于提高路由的效率具有非常重要的作用。因为这些系统中 Flooding 路由终点是资源所在地，而不是资源索引，从而确定式缓存能够提供直接的快捷键。与此不同，结构化的 DHT-P2P 系统中 DHT 路由终点是资源索引，因而要区分开缓存结点标识和缓存结点上的数据项。当缓存结点标识时，采用确定式缓存（如 LRU），由于其无向性，并不能够辅助路由达到全局优化（本节的实验已证实了此点）。而当缓存结点上的数据项时，在路由过程中不可直接得到此数据项，需要额外的传送操作，其开销将是非常大的，而且作用有局部性，它的全局优化性能并不确定。因此，在 DHT-P2P 系统中，往往采用确定缓存用于减少"热点"问题[24]。

值得注意的是，近来提出了一些具有常量度的 DHT 新系统构造，能够采用常量的状态 $O(1)$ 达到 $O(\log_2 n)$ 的路由效率，如 Koorde[25]、Viceroy[14]等。一方面，它们的工作与本节一样，都是考虑常量状态的低度网络下，达到优化的路由效率。另一方面，相比它们的工作，本节的工作具有以下两个性能优点：

(1) 它们必须重构整个网络建立新的系统，而不能够兼容原有老系统。本节工作则是在原有系统下的性能改进，而这种向下兼容性往往是为现实所需要的。

(2) 它们工作于特别的拓扑构造，如 de Bruijn、Butterfly，应用性能很大程度局限于拓扑的特性。本节工作结合了小世界模型理论，是一种普适技术，并不特别强调拓扑的特性，因而具有广泛的应用领域。本节采用的技术方法可以进一步作用于这些特殊的拓扑之上。

3.3 Chord 最优路由

Chord 是由 2^b 个结点构成的一个环，环上的每个结点保存该结点后面 $2^0, 2^1, 2^2, \cdots, 2^{m-1}$ 位共计 $m(m<b)$ 个后续结点的指针信息。由于主机间的 TCP 连接是双向的，因此 Chord 环实际上也是双向的[26]。但是标准的 Chord 协议路由只用了边的一个方向。这里首先给出 Chord 协议路由的形式化描述，将路由过程抽象成一个整数由一个数列受限的线性表示问题。然后，利用 Chord 的双向边来寻求路由表结构的优化，分析并提出 Chord 协议在满环情况下的最优路由表结构，给出基于最优路由表结构的路由算法，并证明在 Chord 满环情况下三倍数路由表结构是最优路由表结构。接下来，针对于实际中 Chord 环的非满环特性，提出了多种路由表构造算法，实验证明该构造比原 Chord 的路由效率更高。

P2P 网络的关键问题是对象的定位和路由[27]。在这方面有很多的研究工作正在进行，总体来看，主要分为两种方式：一是非结构化路由，二是结构化路由。非结构化路由有 Gnutella[28]、Freenet[29]等。每个结点仅维护邻居路由信息，搜索采用洪泛方法，搜索的范围

用 TTL 参数来控制，很显然搜索在超出一定范围后不能进一步扩展。结构化路由有 Tapestry[30～33]、Pastry[34]、Chord[27]等，它们都基于 DHT (Distributed Hash Table)[5]技术，路由表具有良好的结构性，查找具有确定性，它具有较好的可扩展性能。分布式哈希算法是适用于规模可扩展网络的新一代路由算法，该算法利用小世界原理，在网络中的每个结点只存储部分其他结点的信息，每个结点及文件都通过预定的哈希函数映射到一个 Overlay 层空间的一个 ID 上，具体的路由由多个结点协作完成。DHT 方法的共同特征是将对象空间映射到虚拟名字空间，在虚拟名字空间中均匀分配映射对象，从而方便地实现对象到结点的映射。结构化路由中每个结点维护一个按一定的方法构造的结点列表，用来改进路由效率。

与现有的工作相比，Chord 算法具有简单性、可验证性和可扩展性等优点。Chord 算法的路由跳数和网络直径优于 CAN 算法，结点加入过程和维护开销优于 Tapestry 算法和 Pastry 算法，因此本节主要基于 Chord 算法进行研究。

我们注意到，由于主机间的 TCP 连接是双向的，因此，实际上 Chord 拓扑是双向的，但是标准的 Chord 协议只用了边的一个方向，正是因为 Chord 的双向性，使得原来的路由表(Finger Table)中的许多项变得冗余起来，因为 3＝4－1 并且 3＝2＋1，即可以从两个方向来逼近目标接点。因为在双向 Chord 的路由表中，不仅保存了正向的路由表，而且还保留了反向的路由表。在这种情况下，如何最优化路由表结构，是本节研究的主要内容。

3.3.1　Chord 协议

Chord 算法[35]是由 MIT 提出的一种分布式查找协议，提供一种基于 DHT 的有效分布式查找服务。

Chord 的核心在于提供了一个快速的分布式哈希计算的功能，把结点的 ID 和文件映射到存储它们索引的结点上去。Chord 利用连续哈希，一方面，这样可以基本实现负载平衡(每一个结点基本上能够接受数量差不多的值)，另一方面，当第 n 个结点加入(离开)网络时，只有 $O(1/n)$部分值被移到了不同的地方，这显然是用最小的代价维护了负载均衡。

另外，Chord 协议可以使得每一个结点并不需要知道所有的其他结点的位置，改进了连续哈希的可扩展性。一个 Chord 结点只需要知道一小部分关于其他结点的路由信息。因为这些信息都是分布式的。在稳定状态下，在一个有 n 个结点的系统中，每个结点只需要维持 $O(\log n)$条关于其他结点的信息。进行搜索也只需要 $O(\log n)$条信息。每次当结点加入或者离开系统时，Chord 都必须更新路由信息，但是基本上每次最多产生 $O(\log^2 n)$条信息。

例如，在 Chord 协议中，该算法中每个结点和文件都映射到 160 位的标识。该标识通过对文件名或结点 IP 哈希得到，结点和文件 ID 在一维空间上构成一个环，文件关键字存储在文件 ID 的后继结点上。

图 3-22 为 Chord 逻辑环及其路由表示意图，图 3-23 所示为一个拥有三个结点 0、1 和 3 的 Chord 环实例。

图 3-22　Chord 的逻辑环及其路由表项

Prasanna Ganesan 和 Gurmeet Singh Manku[26]对 Chord 的路由表进行了研究，该文章的主要思想是鉴于 TCP 的网络连接的双向性，进而分析最优路由表构建。该文章将最优路由表问题转化成一个二进制减法问题(Binary Subtraction Problem)，具体为：给定 $b \geqslant 1$，给定一个整数 $d(0<d<2^b)$，

要确定一个整数对(d',d''),要求 d' 和 d'' 均为 b 位的整数,使得 $d=d'-d''$ 或者 $2b-d=d''-d'$ 成立。在这个情况下,最小化 $H(d')+H(d'')$,其中,函数 $H(x)$ 为 x 的海明范数(Hamming Norm),海明范数也就是整数 x 的二进制表示中 1 的个数。解出这个二进制减法问题的最优解,也就对应了原 Chord 的最优路由。

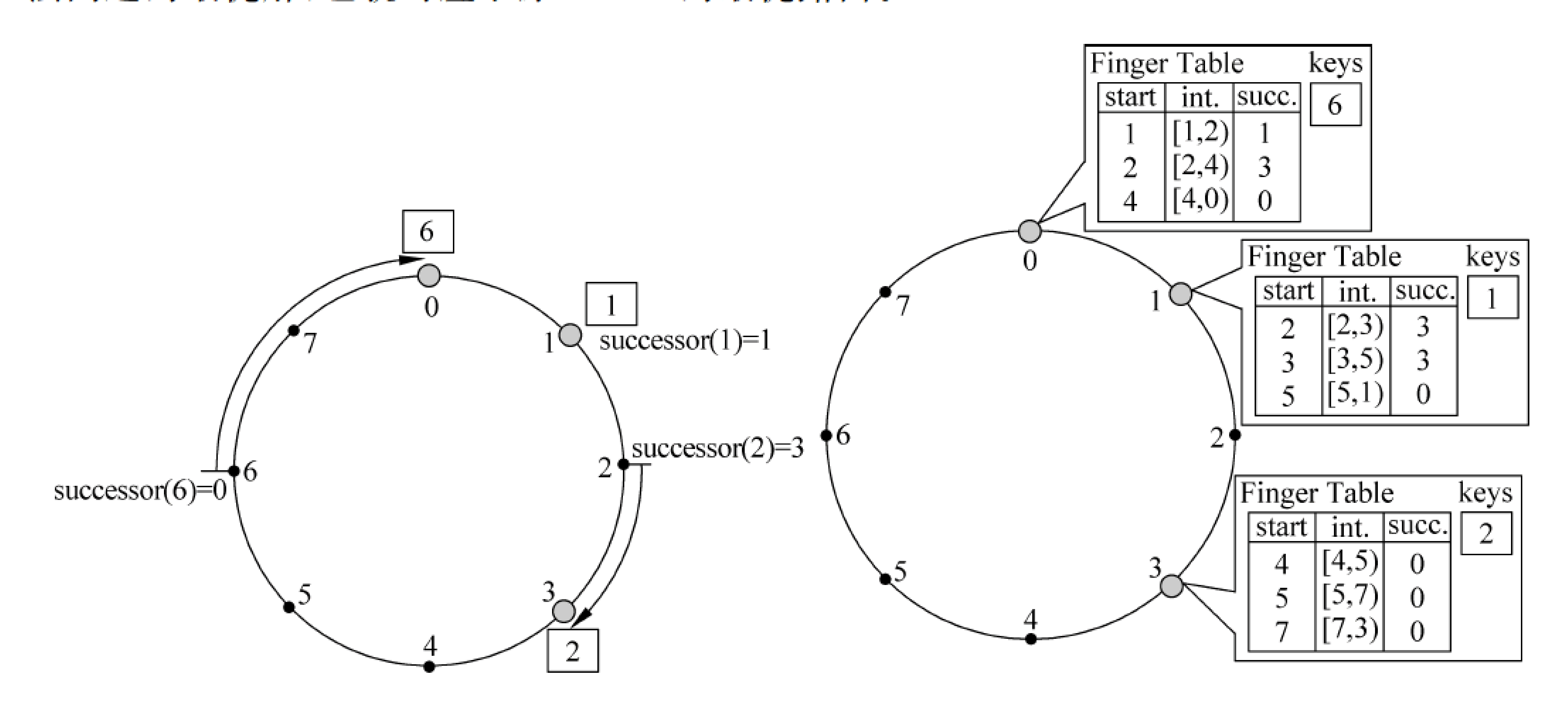


图 3-23　拥有三个结点 0、1 和 3 的 Chord 标识环

3.3.2　问题描述

下面分两种情况进行研究:一种是当 Chord 环为满环情况,另一种是当 Chord 环为非满环的情况。

1. 相关定义

定义 3-8　在 Chord 协议中,每一台机器经哈希都唯一对应 Overlay 层的一个 ID。如果在 Chord 环的 ID 空间中不存在空闲的 ID,即不存在没有被机器哈希到的情况,称这种情况下的 Chord 环为满环。

显然,满环是一种极端情况,在满环情况下,没有结点的进入和退出,故是 Chord 环是静态的。

定义 3-9　每一台机器经哈希都唯一对应 Overlay 层的一个 ID,同时,至少存在一个 ID 没有被主机哈希到。我们称这种情况下的 Chord 环为非满环。

2. Chord 路由的形式化

首先,从数学角度上看 Chord 路由算法可以分为以下步骤:

(1) 计算出源结点和目的结点之间的顺时针的 ID 距离为

$$d=(y-x+2^b)\bmod 2^b$$

(2) 将这个数 d 用二进制表示出来。

(3) 利用该二进制数中的从左到右 1 的位置进行路由。

显然,该路由是单性的,仅仅利用到了 Chord 的单项边,但在实际网络中,由于 TCP 连接是双向的,因此 Chord 环应该是一个无向环。

在 Chord 协议中,任意一个结点 P 仅维护有限个(n 个)其他结点的信息,这些信息构成

该结点的路由表，记为：$F_p=\{a_1,a_2,\cdots,a_n\}$，其中，$a_i$ 表示路由表中的第 i 个路由结点指针所指向的结点的 Overlay 层 ID 与本结点 ID 的模 N 差为 a_i，用 ID(P)来表示结点 P 的 Overlay 层的 ID 号。

将点 P_i 查找 P_j 表示为：$P_i \to P_j$。当 P_i 的 Overlay 层路由表中不包括 P_j 时(即$[\mathrm{ID}(P_i)-\mathrm{ID}(P_j)] \bmod 2^n \notin F^{p_i}$)，$P_i$ 需要从其 Overlay 层路由表中找出一点 $P_k \in F^{p_i}$，将查找任务转交给 P_k，由 P_k 来继续完成查找任务。这是一个逐步逼近的过程。

再抽象一下，P_i 到 P_j 的 ID 距离(模 N)为 m，路由实际上就是用一个数列对 m 的线性表示问题。

总结 Chord 的路由条件(即条件 0)，形式化如下：

路由过程：Chord 标准路由表为$(a_1,a_2,\cdots,a_n)$，路由 $P_i \to P_j$ 结点时，有$[\mathrm{ID}(P_j)-\mathrm{ID}(P_i)] \bmod n=m$，要求:各项的组合之和为 m；路由时选择的前项大于后项。下面分别介绍。

(1) 各项的组合之和是 m。其中，$m=\sum_{i=1}^{n}[(a_i)*s(i)]$，这里 a_i 表示当前结点路由表的第 i 项，$s(i)$为一个可以取 0、+1 和 −1 的函数，m 是 n 个数 $a_1,a_2,\cdots,a_n$ 受约束条件下的线性表示。

(2) 路由时选择的前项大于后项。Chord 是一种贪婪法路由，每一跳都要求尽可能大地逼近目标结点，因此在路由表选择上，要求满足前项大于后项。

3. 数学问题描述

我们将 Chord 的路由过程归结为一个整数的数列表示问题。

问题 1　路由跳数优化问题。

给定一个整数 d 和一组数列$(a_1,a_2,\cdots,a_n)$，要求确定一个向量$(k_1,k_2,\cdots,k_n)$，其中 k_i 的取值可为 0，−1 或 1，使得：

$$d=\sum_{i=1}^{n}(k_i*a_i)$$

并同时满足 $\min\left(\sum_{i=1}^{n}|k_i|\right)$。为了方便下面叙述，记为：$\text{min_hops}=\min\left(\sum_{i=1}^{n}|k_i|\right)$

问题 2　路由表结构的优化问题。

在一组数列$(a_1,a_2,\cdots,a_n)$下，对于不同的整数 d_j，其平均最小路由跳数为 $\text{avg}(\text{min_hops})=\left(\frac{1}{m}\sum_{j=1}^{m}\left(\min\left(\sum_{i=1}^{n}|k_i|\right)\Big|_{d_j}\right)\right)$，显然，avg(min_hops)与$(a_1,a_2,\cdots,a_n)$相关，调整数列$(a_1,a_2,\cdots,a_n)$结构，使得 $\min(\text{avg}(\text{min_hops})|_{(a_1,a_2,\cdots,a_n)})$。

3.3.3　理论分析

首先分析上述的问题 2，即路由表构造优化问题。为了叙述方便，结点的 n 个路由表项中所指向的结点 ID 编号与本结点的 ID 编号差(模 M)分别记为 $a_1,a_2,\cdots,a_n$。直观上来看，首先考虑当路由表大小为 1 时，也即是每个结点仅有唯一的一个路由指针时，在满足条件(0)的情况下，根据这个路由表结构每个结点所能路由的 ID 距离为 $-a_1$、a_1 和 0，为了保证每个结点都能路由到，只有使 $a_1=1$，这时可以路由的范围为$[-1,1]$。

设：$S_k=a_1+a_2+\cdots+a_k(k\geqslant 1)$，假设当路由表大小为 k 时，它所能表示的满足条件

(0)的最大范围为$[-S_k,S_k]$；那么，当路由表大小为 $k+1$ 时，它所能表示的满足条件(0)的最大范围为$[-S_{k+1},S_{k+1}]$；因为第一步可以先左或右路由到 a_{k+1} 处，然后才能利用 a_1，a_2，…，a_k 依次路由到相应结点，因此必须满足 $a_{k+1}=2S_{k+1}=2(a_1+a_2+\cdots+a_k)+1$，如图 3-24 所示。

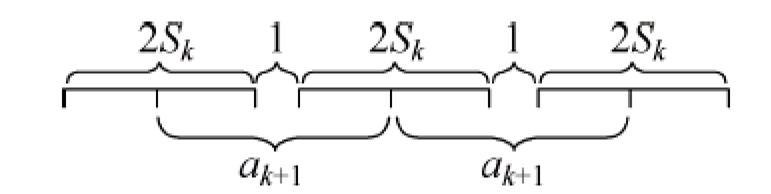


图 3-24　三倍数最优说明示意图

根据分析，容易得出 $a_1=1,a_2=3,a_3=9,\cdots\cdots$归纳法得出最优路由结构为三进制。

引理 3-4　有一组数(共 n 个)，记为 $a_1,a_2,\cdots,a_n$，它们两两不等，即当 $i\neq j$，有 $a_i\neq a_j$ 恒成立，将这 m 个数按从小到大排列成一个数列$\{a(1),a(2),\cdots,a(n)\}$，设 $A(k)=\sum_{i=1}^{k}a(i)$，则：

$$a(1)=1 \text{ and } a(j+1)\leqslant 2A(j)+1,\quad 1\leqslant j\leqslant n-1 \tag{3-30}$$

是该数列中各数的加减法组合可以表示$\{-A(n),\cdots,-1,0,1,2,\cdots,A(n)\}$范围内的任意数字的充要条件。

证明：下面用数学归纳法证明充分性。

首先，我们知道要表示$\{-A(n),\cdots,-1,0,1,2,\cdots,A(n)\}$范围内的任意数字$\Leftrightarrow$表示$\{0,1,2,\cdots,A(n)\}$范围内的任意数字，因为对于数列$\{a(1),a(2),\cdots,a(n)\}$来说，只需改变组合符号就可以等价表示其相反数，因此以下只需证明：式(3-30)是该数列中各数的加减法组合可以表示$\{0,1,2,\cdots,A(n)\}$范围内的任意数字的充分条件即可。

(1) 当 $i=1$ 的时候，$a(i)=1$ 显然成立。

(2) 假设 $i=k$ 时引理充分性成立，即用满足式(3-30)的前 k 个路由表项可以路由的 ID 差范围 $W(k)$为满足 $0\leqslant W(k)\leqslant A(k)$的所有整数。

(3) 当 $i=k+1$ 时，要证明的是可以路由到的 ID 差范围 $W(k+1)$，应为 $0\leqslant W(k+1)\leqslant A(k+1)$范围内的所有整数。现分段讨论如下：

① 对于 $0\leqslant W(k+1)\leqslant A(k)$，显然可以由前 k 个路由表项可以路由到。

② 对于 $A(k)<W(k+1)\leqslant a(k+1)$，只需表示范围为 $0\leqslant W(k)\leqslant A(k)$即可。因为交换向上向下的路由方向，可以产生配合路由表项 $a(k+1)$使用的反向边为 $W(k)'$ 可以是满足$-A(k)\leqslant W(k)'\leqslant 0$ 的所有整数。与路由表项 $a(k+1)$一起使用可以得到 $a(k+1)+W(k)'$，一定可以路由某段连续范围的所有整数。因为 $a(k+1)\leqslant 2A(k)+1$，所以 $a(k+1)-A(k)\leqslant A(k)+1$，因此 $a(k+1)+W(k)'$产生的下限为 $a(k+1)-A(k)$，上限为 $a(k+1)$，所以可以路由到 $A(k)<W(k+1)\leqslant a(k+1)$内的所有 $W(k+1)$。

③ 对于 $a(k+1)<W(k+1)\leqslant A(k+1)$，与②同理可以得到路由的上下限分别为：$a(k+1)+A(k)=A(k+1)$和 $a(k+1)$。因此当 $i=k+1$ 时，定理充分性也成立，由数学归纳法知定理充分性成立。

下面证明必要性。

(1) $i=1$ 时，显然必须有距离为 1 的路由表项；

(2) $i>1$ 时，反证之，如果存在某个 K，使得式(3-30)不成立，即 $2A(k)+1<a(k+1)$，

则路由项 $A(k)+1$ 不能用前面的 $k-1$ 个路由表项路由，又因为 $a(k+1)-A(k)>A(k)+1$ 而不能用 $a(k+1)$ 配合着路由，所以矛盾，因此必要性成立。证毕。

定理 3-5　Chord 环为满环，其上的每个结点的路由表由 n 个互不相同的结点的 ID 号组成，这些 ID 号与本结点 ID 的模 N 差构成一组数(共 n 个)，将这 n 个数按从小到大排列成一个数列 $\{a(1),a(2),\cdots,a(n)\}$，设 $A(k)=\sum_{i=1}^{k}a(i)$，其中 i 从 1 到 k，则

$$a(1)=1 \text{ 且 } a(j+1)\leqslant 2A(j)+1,\quad j=1,2,\cdots,m-1 \tag{3-31}$$

是该数列作为路由表序列可满足条件(0)路由到 $\{-A_m,\cdots,-1,0,1,\cdots,A_m\}$ 范围内的任意结点的充要条件。

证明：根据引理，即可得证。

通过刚才的证明已经证明了路由表构造的最优化问题，现在来考虑在这个路由表构造下的路由优化问题，即问题 1。

推论 3-1　将整数 n 用由序列 $\{a(1),a(2),\cdots,a(m)\}$($m$ 个整数)加减组合来表示，设 $M=\sum 3^{i-1}$，其中 i 从 1 到 m，则有三种情况：

(1) 若 $M<n$，无解；

(2) 若 $M=n$，有唯一的解，$a(j)=3^{j-1}$，$j=1,2,\cdots,m$；

(3) 若 $M>n$，有多组解。

证明：

(1) 因为 $M<n$，所以肯定存在一个不能表达。

(2) 因为 $M=n=\sum 3^{i-1}$，其中 i 从 1 到 m，存在性在前面定理已经给出了，现在给出唯一性证明(反证法)：

假设存在两种不相同的序列 $\{f_1(i)\}$ 和 $\{f_2(i)\}$，其中 $f_1(i)=0,1$，或 -1，$f_2(i)=0,1$，或 -1，其中 i 可取从 1 到 m，使得给定的数字 $d\in\{0,1,2,\cdots,3^{m-1}\}$，可以表示为 $d=\sum_{i=1}^{m}[f_1(i)\times a(i)]$ 并且 $d=\sum_{i=1}^{m}[f_2(i)\times a(i)]$。

由上面两个式子可以得出：

$$\sum_{i=1}^{m}[(f_1(i)-f_2(i))\times a(i)]=0$$

即是说 $\{a(i)\}$ 可以线性表示，但由于 $a(i)=3\times a(i-1)=2\times\sum_{i=1}^{i-1}a(i)+1$，$i=1,2,\cdots,m$，$f_1(i)$ 和 $f_2(i)$ 的取值范围均为 $\{-1,0,+1\}$，所以可以推出 $f_1(i)-f_2(i)$ 的取值范围为 $\{-2,-1,0,+1,+2\}$，$a(i)=2\times\sum_{i=1}^{i-1}a(i)+1>2\times\sum_{i=1}^{i-1}a(i)$，因此在 $f_1(i)$ 和 $f_2(i)$ 的取值范围下不可能使等式成立，故矛盾。序列 $\{f_1(i)\}$ 和 $\{f_2(i)\}$ 相同，唯一性得证。

(3) 若 $M>n$，有多组解，解为满足条件(0)并且 $\sum a(i)=n$，其中 i 取从 1 到 m 的所有整数序列。证毕。

根据以上的定理和推论，可以很容易的求出对于 ID 空间为 n，用 m 个路由表项可以按要求路由到任一指定结点，当 $n=\sum 3^{i-1}$，i 从 1 到 m 的时候，有唯一解 $a(i)=3^{i-1}$，这个正是

最优的路由表结构。

根据推论 3-1 中三倍数路由表表示的唯一性可知，只要能够表达的就是最小的，因此问题 1 的解就变成给定整数的数列受限组合表示问题。

3.3.4 算法描述

一个结点查询 key 值，具体的路由算法的伪代码表示如下：

```
    routing(node_i, key )
{
1:      find(node_i, hash(key));
}

find(node_i, node_j)
{
1:      d = (node_i - node_j)modS;
2:      if( d>= 0 and d<S/2)
3:              NextHop(node_i, d)
4:      else if(d>= S/2 and d<S)
5:              NextHop(node_i, d - S)
}

NextHop(node_i, d)
{
1:      print node_i;
2:      if(d = 0)
3:              return ;
4:      else if( d>0)
5:              k = find(d) in pos_finger_table;
6:              if(|d-pos[k]|<= |d-pos[k+1]|)
7:                      NextHop(node_k, d - pos[k]);
8:          else
9:                      NextHop(node_k+1, d - pos[k+1]);
10:     else
11:             k = find(d) in neg_finger_table;
12:             if(|d + neg[k]|<= |d + neg[k+1]|)
13:                     NextHop(node_k, d - neg[k])
14:             else
15:                     NextHop(node_k+1, d - neg[k+1])
16:     endif;
}
```

在下一跳确定时，通过判断 d 的正负来表明这次跳转的方向，为正则顺时针跳转，为负则逆时针跳转，即为正则在正向路由表中选择，为负则在反向路由表中选择。NextHop 算法中的语句第 2 行表明如果距离 d 为 0，则找到目的结点结束；第 4～9 行表明在正向路由表中查找对应的行进行路由，第 10～15 行表明在反向路由表中查找对应的行进行路由。第 5～9 行和第 11～15 行来确定下一跳距离，判断 d 处于$\{a(1),a(2),\cdots,a(m)\}$中的哪个子段，即找到 k，$1\leqslant k\leqslant m-1$，$a(k)\leqslant d\leqslant a(k+1)$，其中 k 可取 1 到 $m-1$；在三倍数路由表的 Chord 路由中，找出 i，使得 $a(k)<d<\sum a(i)$ for all $i\leqslant k$。

这是找到了下一跳的路由空间，至于具体跳到哪个结点上，需要利用 d 与 $\sum_{i=1}^{k} a(i)$ 的大小，如果 d 大，则要用到 $a(k+1)$，也即正向路由到 $a(k+1)$，然后递归调用 NextHop$(d-a(k+1))$。如果 d 小，则不会用到 $a(k+1)$，仅由$\{a(1),a(2),\cdots,a(k)\}$就可以表示了，也即正向路由到 $a(k)$，然后递归调用 NextHop$(d-a(k))$。

现在来举例说明。满环情况下，图 3-25 显示了一个 ID 环，设 m 为 4。这个环上有 81 个结点，因为是满环，每个结点都对应一台机器，结点 ID0 的正向路由表为：第一个路由指针指向结点 1，第 2 个指向结点 3，第 3 个指向结点 9，第 4 个指向结点 27，即 $a(1)=1,a(2)=3,a(3)=9,a(4)=27$。ID0 希望查找 ID 为 25 的结点，其路由过程如图 3-25 所示。也即给定一个整数 25，将 25 用路由表来线性表示。因为 $9<25<27$，并且 $25>1+3+9$ 所以首先正向路由到 $a(4)$结点，然后调用算法（参数为 $25-27=-2$），$1<2<3$，再判断 $2>1$，因此逆时针跳转到该结点的 $a(2)$结点，然后递归调用算法（参数为$-(2-3)=1$），递归调用直到找到该结点。依次跳的结点序列为：$+27,-3,+1$。其中正号表示顺时针跳转，负号表示逆时针跳转。三倍数路由表结构时 Chord 路由过程举例如图 3-25 所示。

定理 3-6　*在 N 个结点的网络里最优路由表结构下路由的平均路由跳数为 $O(\log_3 N)$。*

证明：假设一个结点 n，要解析一个查询值 K 的后继结点的请求。设 p 是这个值 K 的直属前驱结点。下面分析一下到达结点 p 需要的查询次数。

如果 n 不等于 p，那么 n 就要把这个查询请求交给它自己的路由表里面与 K 最接近的结点。假设结点 p 是在结点 n 的正向（反向类似）路由表的第 i 个间隔中，即 p 在结点 n 的$[(n+3^{i-1}) \bmod S,(n+3^i) \bmod S]$间隔里面，其中 S 是该 Chord 环空间的大小。那么由于这个间隔不是空的（因为包含 p），结点 n 将在这个间隔中找到一个结点 f。n 与 f 之间的距离（中间 ID 的数量）至少是 3^{i-1}，但是 f 与 p 都是在 n 的第 i 个间隔里面，即都在结点 n 的$[n+3^{i-1},n+3^i]$间隔里面，所以它们的距离最多是 3^{i-1}。这也就意味着 f 离 p 比 n 离 p 更加近，而且 f 到 p 的距离最多是 n 到 p 距离的 1/3。

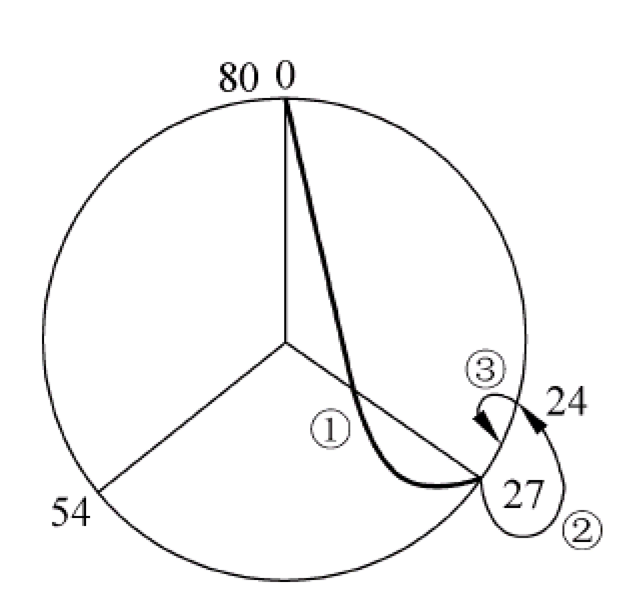


图 3-25　三倍数路由表满环 Chord 路由过程

如果结点每次处理查询与结点 p 的之间的距离都能够减 2/3 的话，那么一开始就能最多有 3^m 了，然后在 m 步之内，这个距离将变为 0，意味着已经到达 p 了。证毕。

3.3.5　Chord 非满环时的分析

在 Chord 非满环情况下，发现 Chord 的路由表中存在着严重的信息冗余，尤其是起始 i 比较小的路由表项，其路由表路由指针指向均相同。

原因是对等网络 Chord 中的结点个数相对于 2^{160} 的巨大的标识空间来说是非常少的，这些点基本上是均匀分布的。当路由表起始的几个路由表项中，由于间隔较小，这使得相邻的两个甚至多个指针指向完全相同，因此造成路由表项重复。我们的思路是利用在 Chord 不满环的情况下，重新构造指针的指向，尽量减少冗余信息，提高查询效率。

在 Chord 非满环情况下的路由过程：Chord 标准路由表为$(a_1,a_2,\cdots,a_n)$，路由从 $P_i\rightarrow$

P_j 结点时，有 $(\mathrm{ID}(P_j)-\mathrm{ID}(P_i)) \bmod n=m$，要求：各项的组合之和是 m，其中 $m=\sum_{i=1}^{n}((a_i+\Delta a_i)\times s(i))$，这里 a_i 表示当前结点路由表的第 i 项，Δa_i 为当前结点的第 i 项的路由修正项，其值为实际值和理论值的差。$s(i)$ 为一个可以取 0、+1 和 −1 的函数，m 是 n 个数 $a_1, a_2, \cdots, a_n$ 受约束条件下的线性表示。

通过分析 $\{\Delta a_i\}$ 的分布发现，修正项在路由表开始几项取值较大，等路由表指针超过第一个实际结点后，各个结点的 Δa_i 开始服从均匀分布，这是因为 Chord 非满环而引起路由表指向的问题。

鉴于 Chord 路由表中存在严重的信息冗余，为了更有效地提高路由效率，采用 4 种算法对现有的 Chord 路由表进行调整。

利用 S/N 来对非满环的 Chord 环进行压缩，构建出新型的路由表结构，对原来的路由表进行改进。其中，S 为 Overlay 层空间大小，N 为对等网络中结点的个数，可以采用估算值。

下面是 5 种重构路由表方法的伪码。

1. 路由表重构方法一

对 Chord 中每个结点的二进制路由表进行重构，但不维护反向路由表，其中 m1 为该重构 Chord 的路由表大小，重构方法如下：

```
finger_table_type finger_table[m1];
OverlayID myID = getOwnOverlayID();
for(i = 0;i < m1;i++)
{
    finger_table[i].start = myID + (S/N) * 2^i;
}
```

2. 路由表重构方法二

对 Chord 中每个结点的二进制路由表进行重构，同时维护反向路由表，其中 m2 为该重构 Chord 的路由表大小，重构方法如下：

```
finger_table_type finger_table[m2];
OverlayID myID = getOwnOverlayID();
for(i = 0;i < m2;i++)
{
    pos_finger_table[i].start = myID + (S/N) * 2^i;
}
```

3. 路由表重构方法三

对 Chord 中每个结点的路由表重构成三倍数路由表，同时维护反向路由表，其中 m3 为该重构 Chord 的路由表大小，重构方法如下：

```
finger_table_type finger_table[m3];
OverlayID myID = getOwnOverlayID();
for(i = 0;i < m3;i++)
{
```

```
    pos_finger_table[i].start = myID + (S/N) * 3^i;
}
```

4. 路由表重构方法四

对 Chord 中每个结点的三倍数路由表重构成非线性指数路由表，同时维护反向路由表，其中 m4 为该重构 Chord 的路由表大小，重构方法如下：

```
finger_table_type finger_table[m4];
OverlayID myID = getOwnOverlayID();
for(i = 0;i < m4;i++)
{
    pos_finger_table[i].start = myID + (S/N) * 3^(10 * log2(i + 1));
}
```

5. 路由表重构方法五

对 Chord 中每个结点的三倍数路由表重构成变倍数路由表，同时维护反向路由表，其中 m5 为该重构 Chord 的路由表大小，重构方法如下：

```
finger_table_type finger_table[m5];
OverlayID myID = getOwnOverlayID();
for(i = 0;i < m4;i++)
{
    pos_finger_table[i].start = myID + (S/N) * (3 - log10(i)/3)^i;
}
```

3.3.6　实验模拟

为了比较各种路由表重构的性能，模拟一个 Chord 网络，对各种优化后的路由算法进行了测试，该实验是在一台 PC 机（CPU 主频为 1.7GHz，内存为 512MB）上完成的。Overlay 层 ID 空间大小设定为 1 000 000，其中有 1000 个结点，让键值变化从 1000 到 10 000，做独立的实验。实验中的每一个结点从系统中获得一组随机的键值，然后测量每一次查询的 Hop 跳数。

1. 满环情况下的实验

图 3-26 为在满环情况下双向 Chord 最优路由表 Bi-Opt-Chord 和标准 Chord 的平均路由跳数与 Chord 环上路由结点的关系图。从图 3-26 中可以看出，和标准 Chord 相比，在满环情况下，最优路由表结构 Bi-Opt-Chord 的路由跳数要少得多。

2. 非满环情况下的模拟实验

对于不同的路由表重构方法，在不同结点个数时做了模拟实验，得到各种重构方法下的平均查询跳数曲线如图 3-27 所示。从图中可以看出，经过重构后的路由跳数均比标准的 Chord 路由跳数少。平均路由跳数最少的是第 4 种构造方式。

对于效果最好的路由表重构方法四，引入一个系数 K 来调整其指数，发现 K 和路由跳数的关系如图 3-28 所示。在 10 000 个网络结点的时候，考虑该系数的变化对路由跳数的影响。可以看到通过调整 K，可以减少网络平均路由跳数，重构方法如下：

```
finger_table_type finger_table[m4];
OverlayID myID = getOwnOverlayID();
for(i = 0;i < m4;i++)
{
    pos_finger_table[i].start = myID + (S/N) * 3^(k * log2(i+1));
}
```

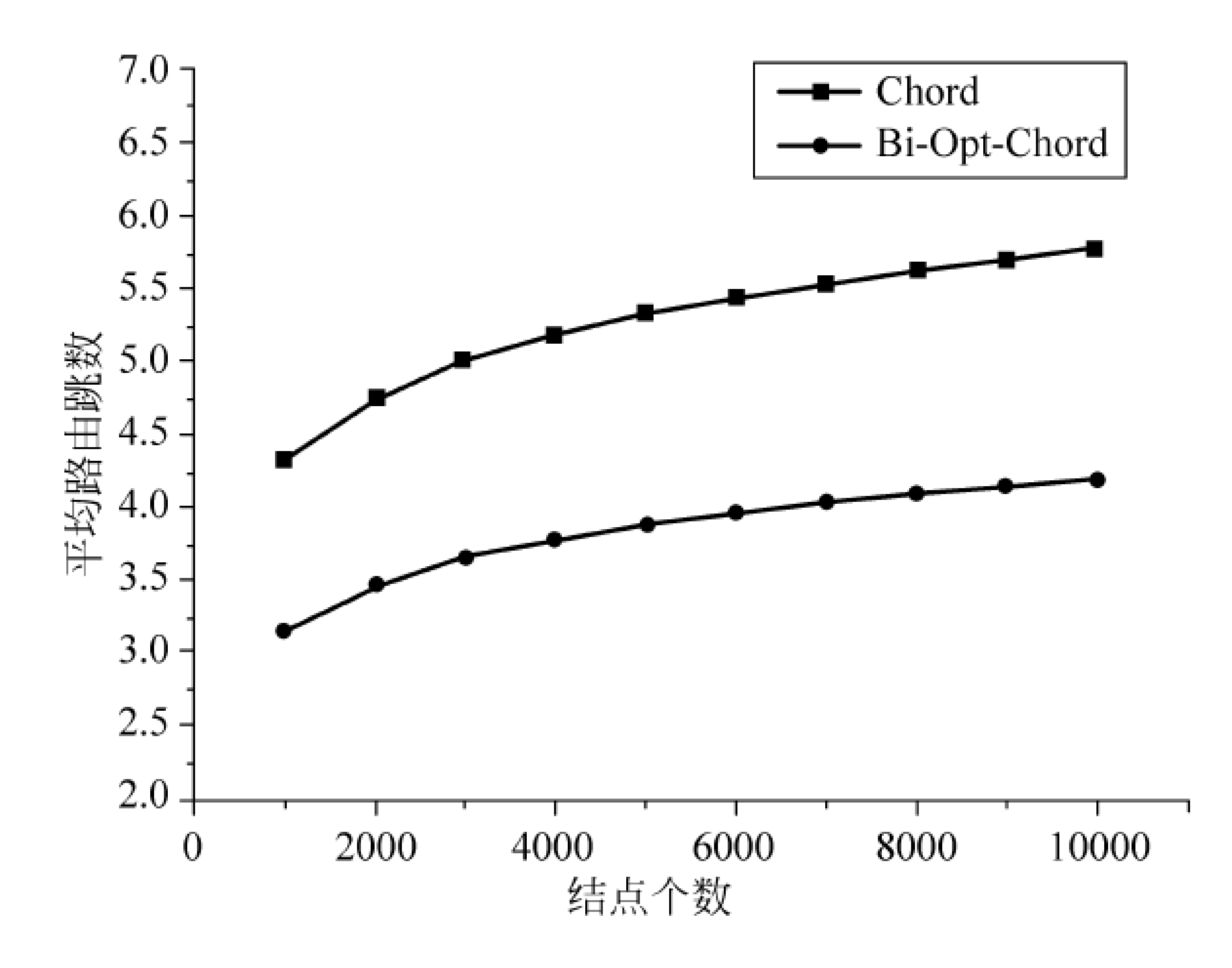


图 3-26　标准 Chord 和 Bi-Opt-Chord 随结点个数变化的路由跳数比较

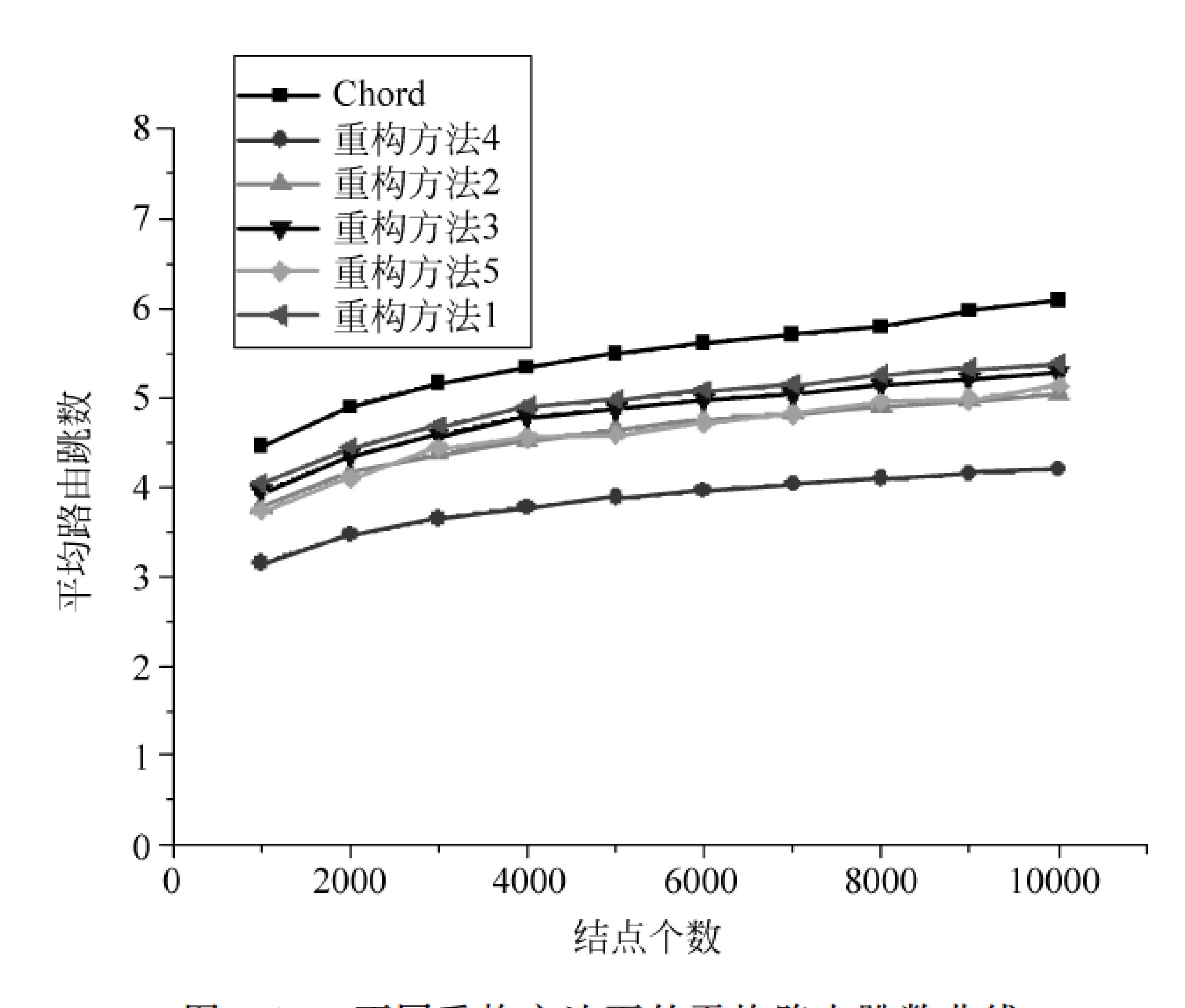


图 3-27　不同重构方法下的平均路由跳数曲线

3.4　本章小结

3.1 节首先说明了结构化 DHT-P2P 系统能够提供高效、可靠的服务，有着巨大的潜在应用前景。同时，指出现实应用中维护 DHT 拓扑，巨大的开销是不可避免的，这就限制了它的应用，尤其是在高度动态的环境下。通过利用 P2P 网络中的会话异构性，3.1 节提出了

一个创新的 SHT 模型来控制维护开销。SHT 模型使用了一个简单但有效的簇技术，主要的技术优点有：

(1) 簇的管理方式是自然而演化的，管理开销很小。由于它不依赖于任何附加的前提条件，所以它可以直接应用于现有的 DHT 算法的改进。

(2) 利用了网络异构特性，即使簇的大小很小，它也能得到一个近似理想的效果，从而对簇的中心结点要求较低，具有良好的现实适用性。

仿真结果表明维护开销可以极大地减少，而查询失败率也在很大程度上也可以减小。因此，采用 SHT 模型在控制拓扑维护开销的同时，进一步提高了 DHT-P2P 系统的稳定性和数据可用性。

3.2 节研究了一种基于小世界理论的概率缓存链技术，并成功应用于 CAN 中，展示了其对于路由性能提高具有的优良特性。它的普遍意义在于能够改进原有的 DHT 系统路由。在这些 DHT 系统的路由表中保留原有的本地短连接，再加入缓存长链，以及在路由过程中实施概率置换策略。由此形成小世界网络，从而改进系统的路由性能。由于缓存长链接是通过本地管理的，这种改进方法对系统的修改程度是很小的，维护也很简单。这种技术特别适合于 DHT 系统在保持低的网络连接度下达到较好的路由性能，并要求有较低的附加网络开销的情况。我们用概率缓存模式来构造小世界网络是 Kleinberg 小世界模型理论在 P2P 网络中的创新应用，具有以下几个特点：

(1) 应用概率缓存模式来构造小世界网络，具有低开销，高容错，缓存具有本地管理性，对于原 DHT 算法修改很少而且相当容易配置等优点，因而具有相当吸引力和可行性的应用前景。

(2) 采用概率缓存模式构造的 DHT 路由具有低状态、高效率的特点，是一种较好状态与效率的折中，并且由于采用概率算法，同时具有较好的容错性和负载均衡。

(3) 概率缓存置换受查询驱动的特性也将使它得益于当前 P2P 系统的一些主流应用如资源共享。在这些应用里是以信息查询为特征的，因而，具有较强的现实意义。

3.3 节鉴于主机间的 TCP 连接是双向的，但是标准的 Chord 协议路由只用了边的一个方向，本节利用 Chord 的双向边来寻求路由表结构的优化。首先给出了 Chord 协议路由的形式化描述，将路由过程抽象成一个整数由一个数列受限的线性表示问题，然后分析并提出了 Chord 协议的最优路由表结构，并给出了基于最优路由表结构的路由算法，并证明了 Chord 在满环情况下三倍数路由表构造是最优路由表结构。而对于 Chord 不满环的情况下对路由表进行重构，引入了多种重构策略，并对其进行了仿真实验，证明在重构后的路由算法切实有效。

参考文献

[1] Balakrishnan H, Kaashoek M F, Karger D, Morris R, Stoica I. Looking up data in P2P systems. Communications of the ACM, 2003. 46(2): 43-48.

[2] Bhagwan R, Savage S, Voelker G. Understanding availability. In Proc. IPTPS, Berkeley, CA, USA, 2003, 2.

[3] Saroiu S, Gummadi P K, Gribble S D. A measurement study of peer-to-peer le sharing systems. In Proceedings of Multimedia Computing Networking 2002 (MMCN), San Jose, CA, USA, 2002, 1.

[4] Ledlie J, Taylor J, Serban L, Seltzer M, Self-Organization in Peer-to-Peer Systems. In the 10th ACM SIGOPS European Workshop. 2002.
[5] Ratnasamy S, Shenker S, Stoica I. Routing Algorithms for DHTs: Some Open Questions. Electronic Proceedings for the 1st International Workshop on Peer-to-Peer Systems (IPTPS'02).
[6] Hazel S, Wiley B. Achord: A variant of the chord lookup service for use in censorship resistant peer-to-peer publishing systems. In Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS'02), MIT Faculty Club, Cam-bridge, MA, USA. 2002.
[7] Wu Z D, Ma F Y, Rao W X. Super-proximity routing in Structured P2P Networks. Journal of Zhejiang University SCIENCE, 2004. 5(1): 16-21.
[8] Liben-Nowell David, Balakrishnan Hari, Karger David. Analysis of the Evolution of Peer-to-Peer Systems. ACM Conf. on Principles of Distributed Computing (PODC), Monterey, CA, 2002, 7.
[9] Mahajan R, Castro M, Rowstron A. Controlling the cost of reliability in peer-to-peer overlays. In IPTPS'03, 2003, 2.
[10] Sen S, Wang J. Analyzing peer-to-peer traffic across large networks. In: Proc. of ACM SIGCOMM Internet Measurement Workshop. 2002.
[11] Xu Z Y, Min R, Hu Y M. Reducing Maintenance Overhead in DHT Based Peer-to-Peer Algorithms. pp. 218-219, Linkpings, Sweden, 2003, 9.
[12] Kleinberg J. The small-world phenomenon: an algorithmic perspective. Cornell Computer Science Technical Report 99-1776, 2000.
[13] Manku G S, Bawa M, Raghavan P. Symphony: Distributed Hashing in a Small World. In Proceedings of the Fourth USENIX Symposium on Internet Technologies Systems (USITS), Seattle, WA, USA, 2003, 3: 127-140.
[14] Dahlia Malkhi, Moni Naor, David Ratajczak. Viceroy: A Scalable Dynamic Emulation of the Butter In Proceedings of the Twenty-First ACM Symposium on Principles of Distributed Computing (PODC), Monterey, CA, USA, 2002, 7: 183-192.
[15] Milgram S. The small world problem, Psychology Today 1967, 1: 61.
[16] Watts D, Dodds P, Newman M. : Identity Search in Social Networks. Science 2002: 296.
[17] Pool I, Kochen M. Contacts influence. Social Networks, 1978, 1: 1-48.
[18] Watts D, Strogatz S H. Collective Dynamics of 'Small-World' Networks, Nature, 1998: 393, 440-442.
[19] Kemeny J G, Snell J L. Finite Markov Chains. New York: Springer-Verlag, 1976.
[20] Overnet, http://www.overnet.com.
[21] Ratnasamy S, Francis P, Hley M, Karp R, Schenker S. A scalable content-addressable network. In Proc. of SIGCOMM, ACM, 2001, 8: 161-172.
[22] Nima Sarshar, Vwani P Roychowdhury. A Rom Structure for Optimum Cache Size Distributed Hash Table (DHT) Peer-to-Peer Design, 2002, arXive: cs. NI/0210010, available at http://www.ee.ucla.edu/~nima/Publications/opt_cache.pdf.
[23] Nitesh Ambastha, Inho Baek, Salil Gokhale, Alexer Mohr. A Cache-Based Resource Location Approach for Unstructured P2P Network Architectures, Department of Computer Science, State University of New York at Stony Brook, Stony Brook, NY, USA, 2003. Available at www.cs.sunysb.edu/~csgsc/grc2003/abstract_inho_2pg.pdf.
[24] Tyron Stading, Petros Maniatis, Mary Baker. Peer-to-Peer Caching Schemes to Address Flash Crowds Crowds, 1st International Peer To Peer Systems Workshop (IPTPS 2002).
[25] Kaashoek M F, Karger D R. Koorde: A Simple Degree-optimal Distributed Hash Table. In Proceedings of the Second International Workshop on Peer-to-Peer Systems (IPTPS), Berkeley, CA, USA, 2003, 2.
[26] Prasanna Ganesan, Gurmeet Singh Manku. Optimal Routing in Chord. Proc. SODA 2004.

[27] Stoica I,Morris R,Karger D,Kaashoek M F,Balakrishnan H. Chord：A scalable peer-to-peer lookup service for internet applications. ACM SIGCOMM,2001.
[28] Gnutella. http://gnutella. wego. com.
[29] LARKE I,SBERG O,WILEY B,HONG T. Freenet：A Distributed Anonymous Information Storage Retrieval System. ICSI Workshop on Design Issues in Anonymity Unobservability,2000,7.
[30] Zhuang S Q,Zhao,B Y,Joseph A D,Katz R H,Kubiatowicz J D. Bayeux：An architecture for scalable fault-tolerant widearea data dissemination. In Proceedings of the 11th International Workshop on Network Operating System Support for Digital Audio Video,ACM,2001,6.
[31] Weatherspoon H,Wells C,Eaton P R,Zhao B Y,Kubiatowicz J D. Silverback：A global-scale archival system. ACM SOSP,2001.
[32] Kubiatowicz J,Bindel D,Chen Y,Eaton P,Geels D,Gummadi R,Rhea S,Weatherspoon H,Weimer W,Wells C,Zhao B. OceanStore：An architecture for global-scale persistent storage. In Proceedings of ACM ASPLOS,ACM,2000,11.
[33] Zhao B,Kubiatowicz K,Joseph. A. Tapestry：An infrastructure for fault-resilient wide-area location routing. Technical Report UCB//CSD-01-1141,University of California at Berkeley,2001,4.
[34] Druschel P,Rowstron A. Pastry：Scalable,distributed object location routing for large-scale peer-to-peer systems. ACM SIGCOMM,2001.
[35] Stoica I,Morris R,Karger D,et al. Chord：A scalable peer-to-peer lookup service for internet applications. ACM SIGCOMM,2001.
[36] Fuhr N,Hartmann S,Knorz,G,Lustig G,Schwantner M,Tzeras K. AIR/X—a rule-based multistage indexing system for large subject fields. In Proceedings of RIAO-91,3rd International Conference “Recherche d'Information Assistee par Ordinateur” . Barcelona,Spain,1991：606-623.

CHAPTER 4

第 4 章 对等网络信息检索

4.1 基于分类器融合的对等网络文档分类算法研究

4.1.1 引言

在对等网络信息检索中关键的一个操作就是如何将结点所拥有文档进行索引(发布),在典型的对等网络中数据资源分布在各个独立的结点上,如何高效地索引这些数据信息资源,对资源的查找、检索会产生重要影响。为了从文档的发布和索引角度来提高效率、效果,采用分类后发布的方法来节省发布开销,而此时分类的精度将直接影响到发布的效果与质量,因此本章对对等网络中文档的分类问题进行研究。分类算法一方面可以用在文档发布上,另一方面可以用在用户的模式确定上。文本分类的精度将直接影响分类发布的效率和用户模式的确定,在本章中,首先给出常见的分类算法,然后对 SVM 进行改进,提出了 SLMBSVM 模型,接着针对于现有分类算法分类精度不高的问题,利用分类器融合的思想将 SLMBSVM 和 kNN 相结合,提出了 m-SLMBSVM-kNN 分类算法。

4.1.2 文本分类问题描述

简单地说,自动文本分类系统的任务是:在给定的分类体系下,根据文本的内容自动确定文本关联的类别。它可以形式化的描述为:已知一个文档集合 $D=\{d_1,d_2,\cdots,d_n\}$ 以及预定义类集合 $C=\{c_1,c_2,\cdots,c_m\}$,文本分类就是要根据每个文本文档 d_i 的特征信息,由计算机自动地为序偶 $<d_i,c_j>\in D\times C$ 赋一个二元布尔值(True 或 False)的过程。这里,如果将布尔值 True 赋予给序偶 $<d_i,c_j>$,就意味着将文本文档 d_i 分配到类别 c_j 中;相反的,如果将布尔值 False 赋予给序偶 $<d_i,c_j>$,就意味着文本文档 d_i 不属于类别 c_j。而如果在单类别分类中,它还可以简化地表示为一个函数映射,即 $\Phi:D\mapsto C$,其表示把文档确定地分为某一类。也即,文本分类这个映射过程可以是一一映

射，也可以是一对多的映射。通过给定的标识样本的信息，找出文本分类的映射规则，当遇到新文本时，根据该规则，可以确定新文本的类别。

4.1.3　文本分类的一般流程

通常情况下，构造自动文本分类器需要经过文本（文档）表示、特征选择、训练学习机器、性能评价这几个阶段，各阶段之间还可能存在反馈指导环节。图 4-1 描述了从机器学习的角度构造自动文本分类器的一般流程。

所谓的向量空间模型就是给定一个文本文档 $d=d(w_1,w_2,\cdots,w_r)$，项 t_k 可以在文档的不同位置重复出现。为了简化问题，通常不考虑项 t_k 在文档中出现的先后次序并要求项 t_k 互异，这时可以把 $t_1,t_2,\cdots,t_r$ 看成一个 r 维的坐标系，而 $w_1,w_2,\cdots,w_r$ 为相应的坐标值，这样 $d(w_1,w_2,\cdots,w_r)$ 可被看成是 r 维空间中的一个向量，称 $d(w_1,w_2,\cdots,w_r)$ 为文档 d 的向量表示，如图 4-2 所示。

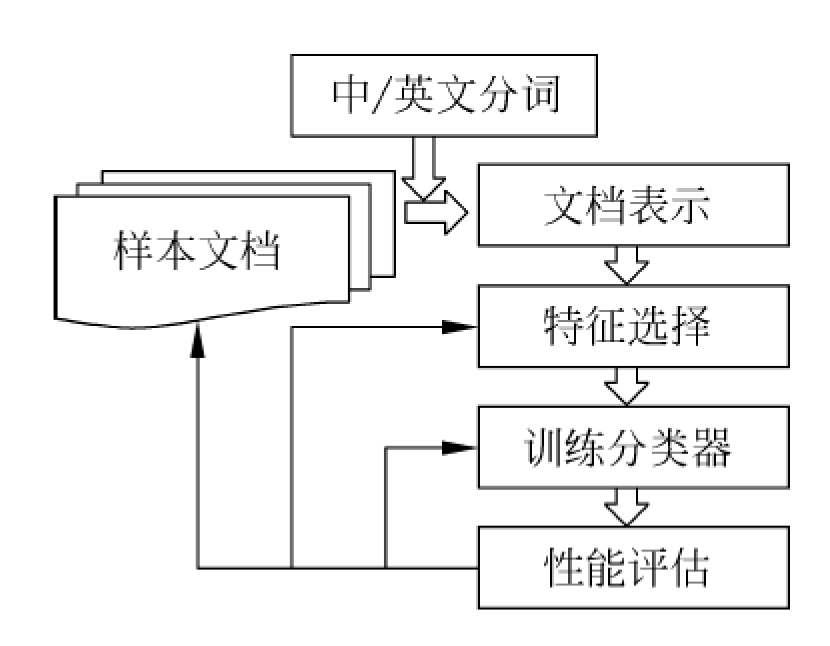


图 4-1　自动文本分类流程

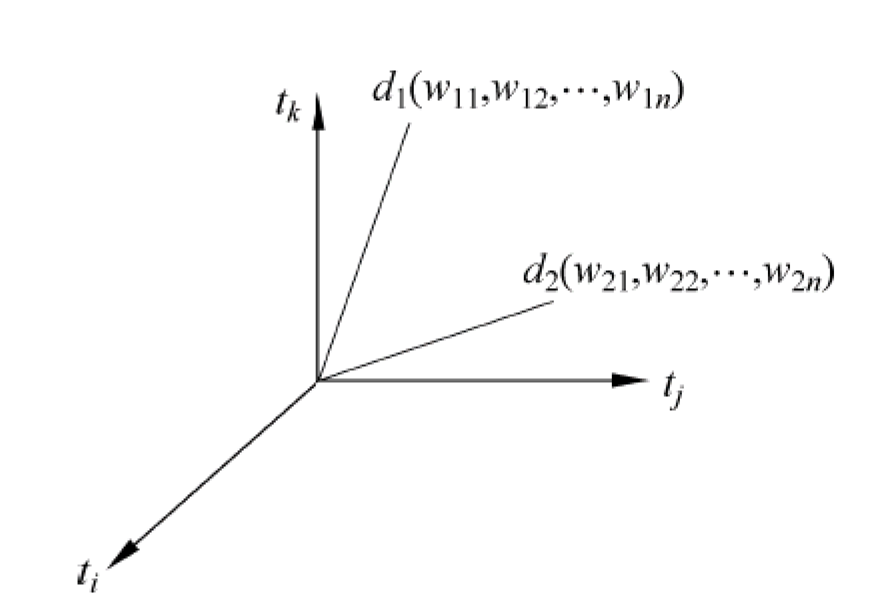


图 4-2　向量空间模型

1. 分词

与英文分词相比，中文分词难度较大。现有的中文分词算法可分为三大类：基于字符串匹配的分词方法、基于理解的分词方法和基于统计的分词方法。基于字符串匹配的分词方法是按照一定的策略将待分析的汉字串与一个词典中的词条进行配，若在词典中找到某个字符串，则匹配成功（识别出一个词）。按照扫描方向的不同，串匹配分词方法可以分为正向匹配和逆向匹配；按照不同长度优先匹配的情况，可以分为最大（最长）匹配和最小（最短）匹配；常用的几种机械分词方法有正向最大匹配法、逆向最大匹配法和最少切分方法。还可以将上述各种方法相互组合，例如，可以将正向最大匹配方法和逆向最大匹配方法结合起来构成双向匹配法；基于理解的分词方法是通过让计算机模拟人对句子的理解，达到识别词的效果，其基本思想就是在分词的同时进行句法、语义分析，利用句法信息和语义信息来处理歧义现象。目前基于理解的分词系统还处在实验阶段。基于统计的分词方法只需对语料中的字组频度进行统计，不需要切分词典，因而又称无词典分词法或统计取词方法。这种方法也有一定的局限性，会经常抽出一些共现频度高、但并不是词的常用字组。实际应用的统计分词系统都要使用一部基本的分词词典（常用词词典）进行串匹配分词。

2. 文本的表示

构造文本分类器的首要任务就是将文本文档性质（文本特征）定量地表示出来，从而将实际的文本文档映射成文本分类器及其构造系统可以处理的数据。

传统 IR 领域中文本分类一般考虑文档的内容，基于“术语-空间”（如向量空间模型 VSM）表示，即将文档表示成权重向量，每个权重向量表示一个特征项的权重，通过计算待分类文本权重向量与预定义类权重向量的相似度决定待分类文本属于哪个类。权重向量的计算一般采用 $tf \times idf$ 方法。

在确定项的权重时，需要遵循的一个基本原则是：为较重要的项赋予较大的权重，以强调它们在文档中的重要程度。目前主要确定项的权重的方法通常采用统计方法，即根据文本的统计信息（如词频）来确定项的权重。

TF（Term Frequency）表示一个 Term 与某个 Document 的相关性。公式为这个 Term 在 Document 中出现的次数除以该 Document 中所有 Term 出现的总次数.

$$w_i = \frac{\mathrm{TF}(t_i, d)}{\sqrt{\sum_j \mathrm{TF}(t_i, d)^2}}$$

其中，j 为文档 d 中 Term 的个数。

TF-IDF（Term Frequency-Document Inverse Frequency）的概念就是一个特定条件下、关键词的概率分布的交叉熵（Kullback-Leibler Divergence）。

$$w_i = \frac{\mathrm{TF}(t_i, d)\log\left(\frac{|D|}{DF(t_i)}\right)}{\sqrt{\sum_i \left[\mathrm{TF}(t_i, d)\log\left(\frac{|D|}{DF(t_i)}\right)\right]^2}}$$

其中，j 为文档 d 中 Term 的个数。

向量空间模型已经在信息检索、文本分类以及其他相关领域得到了广泛而成功的应用，本书也统一使用这一模型表示文本文档的特征。

3. 特征选择

对于一般的文本文档集合来说，用以表示文本特征的项通常可以达到几千个甚至几万个，对应到向量空间模型，每个文本特征向量的维数将高达几千甚至几万维。对于许多机器学习算法来说，处理如此高维的数据是十分困难的，甚至是不可能的。因此，对文本特征加以选择，以降低文本特征向量空间的维数是必要的。所谓特征选择就是从最初的 r 个特征中选取 $r'(r'<r)$ 个特征，而这 r' 个特征可以更为简洁地表示文本文档的内容。如常用的过滤停用词和取词根的方法，实际上就是特征选择，只不过比较简单。其他主要的特征选择标准还有：DIA 关联因子（DIA Association Factor）[1]、卡方检验（Chi-Square Test）[2,3]、NGL 系数（NGL Coefficient）[4]、信息增益（Information Gain）[5]、互信息（Mutual Information）[6]等。

4. 训练分类器

自动文本分类的过程一般来说包括两个步骤：一是训练过程，即将训练样本（已标识为某一类别的文档）通过选定的特征表示方法表示成有监督学习算法可以直接使用的特征向量序列，然后学习算法（即分类算法）对这个序列进行学习，生成分类器；二是经过训练过程后，系统就可以用于对实际文档进行分类，这就是测试过程，即用构造好的分类器将未知类别的测试文档标识为某一类别，实现自动分类的过程。

5. 评估方法

在信息检索研究领域，最常用的评估指标为基于判决损失的度量方法。主要包括错误

率(Error Rate,记为 Err)、查准率(Precision,记为 Pre)、查全率(Recall,记为 Rec)等。

查准率是所有判断的文本中与分类标号结果一致的文本所占的比例：

$$\text{查准率(Pre)} = \frac{\text{分类的正确文本数}}{\text{实际分类的文本数}}$$

查全率是标号分类结果应有的文本中分类系统结果一致的文本所占的比例：

$$\text{查全率(Rec)} = \frac{\text{分类的正确文本数}}{\text{应有文本数}}$$

查准率和查全率反映了分类质量的两个不同方面,但由于在发布过程中主要考虑的是分类类别的差错与否,所以,在这里主要考虑分类精度问题。

将两类错误率分别看待,采用正样本的识别率(Pos_Error_Rate)和负样本的识别率(Neg_Error_Rate)以及整体错误率(Total_Error_Rate)来评价两类别分类器的性能。

$$\text{Pos_Error_Rate} = \frac{FN}{TP + FN}$$

$$\text{Neg_Error_Rate} = \frac{FP}{TN + FP}$$

$$\text{Total_Error_Rate} = \frac{FN + FP}{TP + FN + TN + FP}$$

其中,TP 是正类样本并被正确分类的样本个数；TN 是负类样本并被正确分类的样本个数；FP 是负类样本并被错分为正类的样本个数；FN 是正类样本并被错分为负类的样本个数。

4.1.4　现有的文本分类算法简介

文本自动分类(ATC)作为 IR 领域中的重要的部分,已经受到越来越多研究者的重视。目前出现的 ATC 方法主要有以下几种。

1. 简单向量距离分类法

该方法的分类思路十分简单,根据算术平均为每类文本集生成一个代表该类的中心向量,然后在新文本来到时,确定新文本向量,计算该向量与每类中心向量间的距离(相似度),最后判定文本属于与文本距离最近的类。具体思路是：首先计算每类文本集的中心向量,然后计算新到达的文本特征向量和每类中心向量间的相似度,将文本分到相似度最大的那个类别中。

2. 贝叶斯分类(Bayesian Classification)

目前,在经典贝叶斯理论的基础上提出的朴素贝叶斯(Naïve Bayesian)、贝叶斯网络(Bayesian Network)以及贝叶斯神经网络(Bayesian Neural Network)等方法,已经在很多领域上得到了成功的应用。Mitchell[7]将朴素贝叶斯应用到文本分类中；Lewis 和 Ringuette[8]还有 Moulinier[9]各自独立使用朴素贝叶斯方法对路透社文档测试集作了分类效果评估。

贝叶斯分类的基本思路是计算文本属于类别的概率,文本属于类别的概率等于文本中每个词属于类别的概率的综合,比较新文本属于所有类的概率,将文本分到概率最大的那个类别中。

3. 决策树分类

决策树分类是以实例为基础的归纳学习算法,它着眼于从一组无次序、无规则的事例中

推理出决策树表示形式的分类规则。其中最著名就是 ID3 算法。这一经典算法的基本思想是香农的信息论。ID3 本质上是贪婪算法,使用了基于熵的信息增益来进行分类过程中的属性选择。在 ID3 的基础上,又提出了改进的 C4.5 和 C5 方法[10],以及 CART 和 Assistant 决策树方法。

Moulinier 和 Ganascia[9]、Quinlan[10]、Mitchell[7] 分别将决策树应用到文本分类中;Lewis 和 Ringuette[6] 在路透社文档集上作了决策树分类的效果评估;Moulinier[9] 使用 C4.5 算法对路透社文档集作了分类效果评估。1999 年,美国 Carnegie Mellon 大学的 Kamal Nigam 和 Just 研究中心的 Andrew McCallum 等人提出了一种基于最大熵的文本分类方法,并且与贝叶斯系列分类方法进行了评估比较。

4. 神经网络(Neural Network)

早在 1995 年,Wiener 等就将神经网络应用于文本主题的自动识别[11];1997 年,Ng 等人在 ACM SIGIR 上对神经网络在文本分类上的应用作了综述[12];2000 年,新加坡国立大学的 Ji He 等人将自适应共振联想映射模型(Adaptive Resonance Associative Map,ARAM)应用于 Web 中文文档自动分类中。以上三个研究小组在研究中都使用路透社测试文档集进行了评估。美国施乐公司(Xerox)的 PARC 研究小组开发的 Nnet. PARC 系统[11] 在文本分类中对每个类别分别使用独立的神经网络进行非线性映射,他们也使用路透社测试文档集进行了评估。2002 年,美国俄亥俄大学的 Miguel E. Ruiz[13] 等发表了他们的成果,他们提出了一种专家网络的分级混合模型(Hierarchical Mixture of Experts Model,HMEM)。在该模型中,他们使用分而治之的原则,得到一组分级的神经网络,从而获得最终的文本分类器。这实际上是一种通过神经网络集成来扩展泛化能力的思想。

5. 线性最小二乘拟合(Linear Least Squares Fit,LLSF)

本质上这是一种在输入输出之间的映射方法。在该方法中,训练数据集被表示为输入输出向量对。其中输入是基于传统的向量空间模型表示,输出使用了带二进制权重的类别表示,然后使用多变量回归技术作最小二乘拟合,得到计算结果。Yang Y. 和 Chute[14] 首先提出该方法。

6. 粗糙集(Rough Set)

十余年来粗糙集理论已经得到广泛的研究与应用,在对高维数据的自动分类上,粗糙集有其独到之处。其主要的特点就是可以将高维文本模型通过 Rough 集属性约简(RSAR)转化为包含少量前提条件的规则组成的规则库。

英国 Edinburgh 大学的 Alexios Chouchoulas 教授等人基于粗糙集理论对文本分类进行了研究。在他们实现的系统中,对电子邮件的自动分类进行了测试,并比较了在选取不同训练集的情况下规则库的分类能力。可以认为,文本分类很大程度上依赖于获取描述文件的关键词集合。可以利用 Rough 集方法来降低关键词集合的维数,而又保持关键词集合中足够的信息。采用粗糙集技术是有效的,而且与具体语种无关。这既是优点也是缺点,在处理大规模文本时,如果能利用一些相关的先前的经验知识,必然能提高分类效率和精度。

7. 支持向量机(Support Vector Machine,SVM)

现有的很多分类学习算法一般是基于传统统计学理论的。传统统计学研究的是样本数

目趋于无穷大时的渐近理论，但在实际问题中，样本数往往是有限的，造成一些理论上很优秀的学习方法在实际中却表现得不尽如人意。因此国内外有关学者采用了基于支持向量机SVM(Support Vector Machine)的分类方法[15]，它的理论基础是研究小样本统计估计和预测的统计学习理论。已有的实验表明在文本分类中SVM方法具有较好的分类精度。

支持向量机的基本思想是使用简单的线性分类器划分样本空间。对于在当前特征空间中线性不可分的模式，则使用一个核函数把样本映射到一个高维空间中，使得样本能够线性可分。

1997年，德国的Joachime首次将SVM成功应用于大规模文本自动分类中[16]，它清楚地表明，以小样本空间学习为出发点的SVM同样可以应用于大规模样本集。美国Stanford大学的Simon Tong和Daphne Koller[17]将基于缓冲池的主动学习算法(Pool-based Active Learning)引入文本分类应用中。他们使用的测试数据集分别是路透社测试文档集以及Ken Lang的新闻组测试文档集；Greg Schohn与David Cohn[18]同样就主动学习与支持向量机的结合问题做了研究。

4.1.5　支持向量机原理

支持向量机主要用于解决二分类模式识别问题。其基本思想可用图4-3所示的两维情况说明。图中，圆点和方点代表两类样本，H 为分类线，H_1、H_2 分别为过各类中离分类线最近的样本且平行于分类线的直线，它们之间的距离称为分类间隔(Margin)。所谓最优分类线就是要求分类线不仅能将两类正确分开(训练错误率为0)，而且使分类间隔最大。分类线方程为 $\boldsymbol{x}\cdot\boldsymbol{w}+b=0$，可以对它进行归一化，使得对线性可分的样本集$(\boldsymbol{x}_i, y_i)$，$i=1,\cdots,n$，$\boldsymbol{x}\in R^d$，$y\in\{+1,-1\}$，满足

$$y_i[(\boldsymbol{w}\cdot\boldsymbol{x}_i)+b]-1\geqslant 0,\quad i=1,\cdots,n \tag{4-1}$$

此时分类间隔等于 $2/\|\boldsymbol{w}\|$，使间隔最大等价于使 $\|\boldsymbol{w}\|^2$ 最小。满足式(4-1)且使 $\frac{1}{2}\|\boldsymbol{w}\|^2$ 最小的分类面称为最优分类面，H_1、H_2 上的训练样本点称为支持向量。

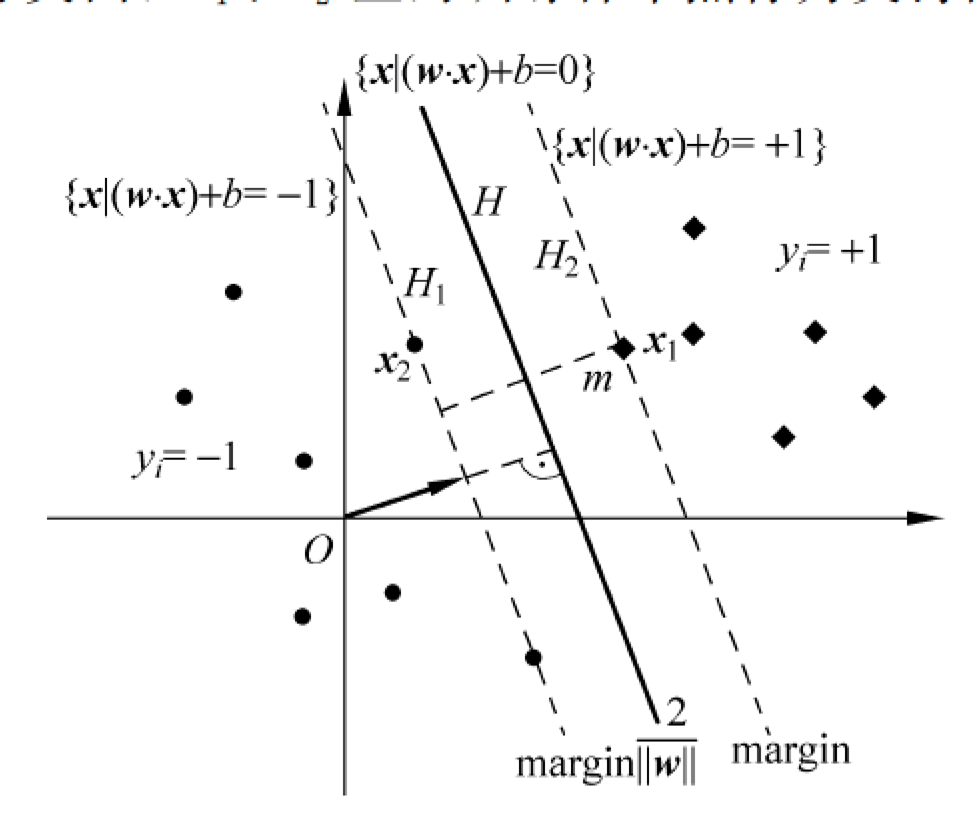


图 4-3　支持矢量机产生的线性决策面

SVM学习的目的就是要构造一个判别函数，将两类模式尽可能正确地区分开来。SVM主要有三种模式：线性可分、线性不可分和曲面分离。所谓线性可分是指存在一个超平面，该平面可以把训练集按类别一分为二。线性不可分是指训练集是线性不可分的，或事

先不知道它是否线性可分，这时通常在超平面上引入一个因子来控制样本偏差与泛化能力之间的平衡。超平面的分类毕竟有限，因此有时需要引入曲面来分割训练集。这三种模式的基本原理基本类似，本文主要是论述在线性可分模式下的 SVM。图 4-4 是支持矢量机从原空间到特征空间的映射关系。

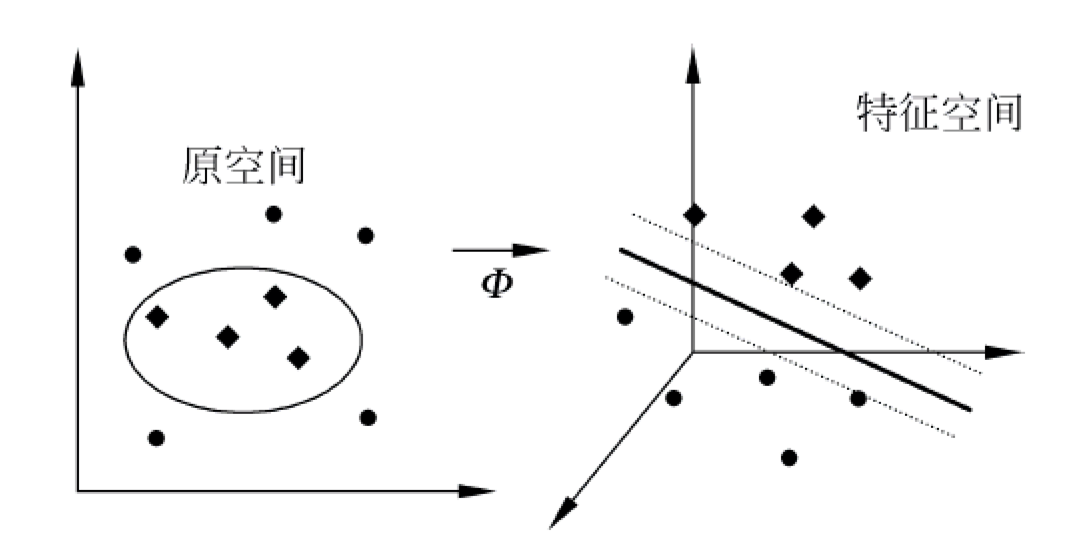


图 4-4　支持矢量机从原空间到特征空间的映射关系

利用 Lagrange 优化方法可以把上述最优分类面问题转化为其对偶问题，即在约束条件

$$\sum_{i=1}^{n} y_i \alpha_i = 0 \tag{4-2a}$$

和

$$\alpha_i \geqslant 0, \quad i = 1, \cdots, n \tag{4-2b}$$

下对 α_i 求解下列函数的最大值：

$$Q(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j (\boldsymbol{x}_i \cdot \boldsymbol{x}_j) \tag{4-3}$$

α_i 为原问题中与每个约束条件即式(4-1)对应的 Lagrange 乘子。这是一个不等式约束下二次函数寻优的问题，存在唯一解。容易证明，解中将只有一部分(通常是少部分)α_i 不为零，对应的样本就是支持向量。解上述问题后得到的最优分类函数是

$$f(\boldsymbol{x}) = \operatorname{sgn}\{(\boldsymbol{w} \cdot \boldsymbol{x}) + b\} = \operatorname{sgn}\left\{ \sum_{i=1}^{n} \alpha_i^* y_i (\boldsymbol{x}_i \cdot \boldsymbol{x}) + b^* \right\} \tag{4-4}$$

式中的求和实际上只对支持向量进行。b^* 是分类阈值，可以用任一个支持向量(满足式(4-1)中的等号)求得，或通过两类中任意一对支持向量取中值求得。

对非线性问题，可以通过非线性变换转化为某个高维空间中的线性问题，在变换空间求最优分类面。在最优分类面中采用适当的内积函数 $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$就可以实现某一非线性变换后的线性分类，而计算复杂度却没有增加，此时目标函数即式(4-3)变为

$$Q(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) \tag{4-5}$$

而相应的分类函数也变为

$$f(\boldsymbol{x}) = \operatorname{sgn}\left(\sum_{i=1}^{n} \alpha_i^* y_i K(\boldsymbol{x}_i, \boldsymbol{x}) + b^* \right) \tag{4-6}$$

这就是支持向量机。

SVM 有两个特性：其中一个特性是可以通过提升数据的维数把线性不可分的训练集

变成为线性可分，另外一个特性是决策平面只是由那些刚好和决策面距离为$\frac{1}{\|\vec{w}\|}$的数据点来决定，这些数据点被称为支持向量。支持向量是训练集中用来生成 SVM 的元素，删除其他数据点不会影响算法的结果(即产生的决策平面(函数)不变)。这个特性是 SVM 与其他分类方法不同之处。

4.1.6　kNN 原理

kNN 是著名的模式识别统计学方法，该方法思想简单，容易实现。kNN 算法本质上是一种聚类算法，通过计算新文档与相邻文档的近似程度来判定所属类别。kNN 算法的基本思想如下：给定一个测试文档，系统在训练集中查找离它最近的 k 个邻居，并根据这些邻居的类别来给该文档的候选类别评分，把邻居文档和测试文档的相似度作为邻居文档所在类别的权重。如果这 k 个邻居中的部分文档属于同一个类别，则对属于该类别的每个邻居的权重求和并作为该类别和测试文档的相似度。通过对候选类别评分的排序，给出一个阈值，就可以判定测试文档的类别。

令 $C=\{c_1,c_2,\cdots,c_n\}$表示类别集，$D=\{\vec{d}_1,\vec{d}_2,\cdots,\vec{d}_m\}$表示训练文档集合，则 kNN 中的决策规则可如下表示：

$$y(\vec{x},c_j)=\sum_{\vec{d}_i\in D}\mathrm{sim}(\vec{x},\vec{d}_i)y(\vec{d}_i,c_j)-b_j$$

其中 $y(\vec{d}_i,c_j)\in\{0,1\}$表示文档$\vec{d}_i$ 是否属于分类 c_j($y=1$ 为是，$y=0$ 为否)；$\mathrm{sim}(\vec{x},\vec{d}_i)$表示测试文档$\vec{x}$和训练文档$\vec{d}_i$ 的相似度；b_j 则是二元决策的阈值。为了方便起见，一般采用两个向量的夹角余弦作为两个文档的相似度。各个分类的阈值 b_j 则是通过训练获得。具体方法是从训练集中抽取部分文档来训练学习这些阈值。

kNN 方法除了简单、易实现以外，与其他分类方法的一个非常突出的不同在于它允许文档可以属于多个分类。

4.1.7　SLMBSVM 模型

与传统的人工神经网络不同，支持向量机是基于结构风险最小化原理，而神经网络是基于经验风险最小化原理。通俗地说，所谓的结构风险最小化就是使错误概率的上界最小化。与神经网络相比，支持向量机不仅结构简单，而且各种技术性能尤其是泛化能力(Generalization)明显有所提高。

显然，传统的 SVM 中两类泛化误差是等同对待的，鉴于实际问题中两类的错误损失往往不是相同的，本节引入了基于损失最小化的支持向量机 SLMBSVMs 的数学模型。这种 SVM 能够针对于两类问题的错误损失的不同，构建整体损失率最小的 SVM。在该数学模型下，系统将以较高的概率正确输出某一类样本，而牺牲另一类的正确率，这是因为该类的错误损失较小。其特点是：①当回答为正样本时，该测试样本将以接近于 1 的概率属于正类；②当回答为负样本时，该测试样本仍有一定的概率属于正类。具体形式如下，SLMBSVMs 的最优决策面要使得下式最小：

$$\min_{w,b,\xi} \frac{1}{2} \| w \|^2 + C_1 \sum_{y_i=1} \xi_i + C_2 \sum_{y_i=-1} \xi_i$$

$$= \min_{w,b,\xi} \frac{1}{2} \| w \|^2 + C\left(n \sum_{y_i=1} \xi_i + \sum_{y_i=-1} \xi_i\right)$$

$$where\ C = C_2, n = C_1/C_2$$

满足：

$$y_i(w \cdot x_i - b) \geqslant 1 - \xi_i, \forall i$$

$$\xi_i \geqslant 0, \ \forall i$$

这里 C_1 和 C_2 是两个反映错误损失的参数，n 为两者的比值。通过引入 Lagrange 乘子来解决这个最优化问题。

$$L(w,b,\xi,\alpha,\beta) = \frac{1}{2} w \cdot w + C\left(n \sum_{y_i=1} \xi_i + \sum_{y_i=-1} \xi_i\right) - \sum_i \alpha_i (y_i (w \cdot x_i - b) - 1 + \xi_i) - \sum_i \beta_i \xi_i$$

其中参数满足以下条件：

$$\frac{\partial L(w,b,\xi,\alpha,\beta)}{\partial w} = w - \sum_i \alpha_i y_i x_i = 0$$

$$\frac{\partial L(w,b,\xi,\alpha,\beta)}{\partial b} = -\sum_i \alpha_i y_i = 0$$

$$\frac{\partial L(w,b,\xi,\alpha,\beta)}{\partial \xi_i} = \begin{cases} Cn - \alpha_i - \beta_i, & \text{if} \quad y_i = 1 \\ C - \alpha_i - \beta_i, & \text{if} \quad y_i = -1 \end{cases}$$

$$= 0$$

代入参数后，变为下面的形式：

$$\min Q(\alpha) = \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j) - \sum_{i=1}^{n} \alpha_i$$

满足：

$$\sum_{i=1}^{n} y_i \alpha_i = 0$$

$$0 \leqslant \alpha_i \leqslant nC, \quad \text{if} \quad y_i = 1$$

$$0 \leqslant \alpha_i \leqslant C, \quad \text{if} \quad y_i = -1$$

首先用一个 R^2 平面上的点来举例说明以上的 SLMBSVM 数学模型。图 4-5 中方点代表类别为负类($Y_i=-1$)而圆点代表正类样本($Y_i=1$)。

SLMBSVM 采用 RBF 作为其核函数 ($\sigma=0.5$)，C 值分别取 100、500、1000，n 值取0.1、0.2、0.5、1、5、10 时其实验结果如图 4-6 和图 4-7 所示。其中，两类别的分类器性能指标的评价采用正样本的识别率(Pos_Error_Rate)和负样本的识别率(Neg_Error_Rate)以及整体错误率(Total_Error_Rate)来评价。

从图 4-6 和图 4-7 中可以看出，随着 n 的增大，会引起 Pos_Error_Rate 的下降和 Neg_Error_Rate 的增加，这表明调节 n 可以取得一个比传统的 SVM 更低的整体错误率。另外，选择合适的 n 值，可以调节当算法输出某个类别时的正确率。

图 4-5 为随 n 和 C 的取值不同 SLMBSVM 的分类决策面的变化图。表 4-1 列出了 SLMBSVM 模型关于泛化错误率和整体泛化损失的实验数据。

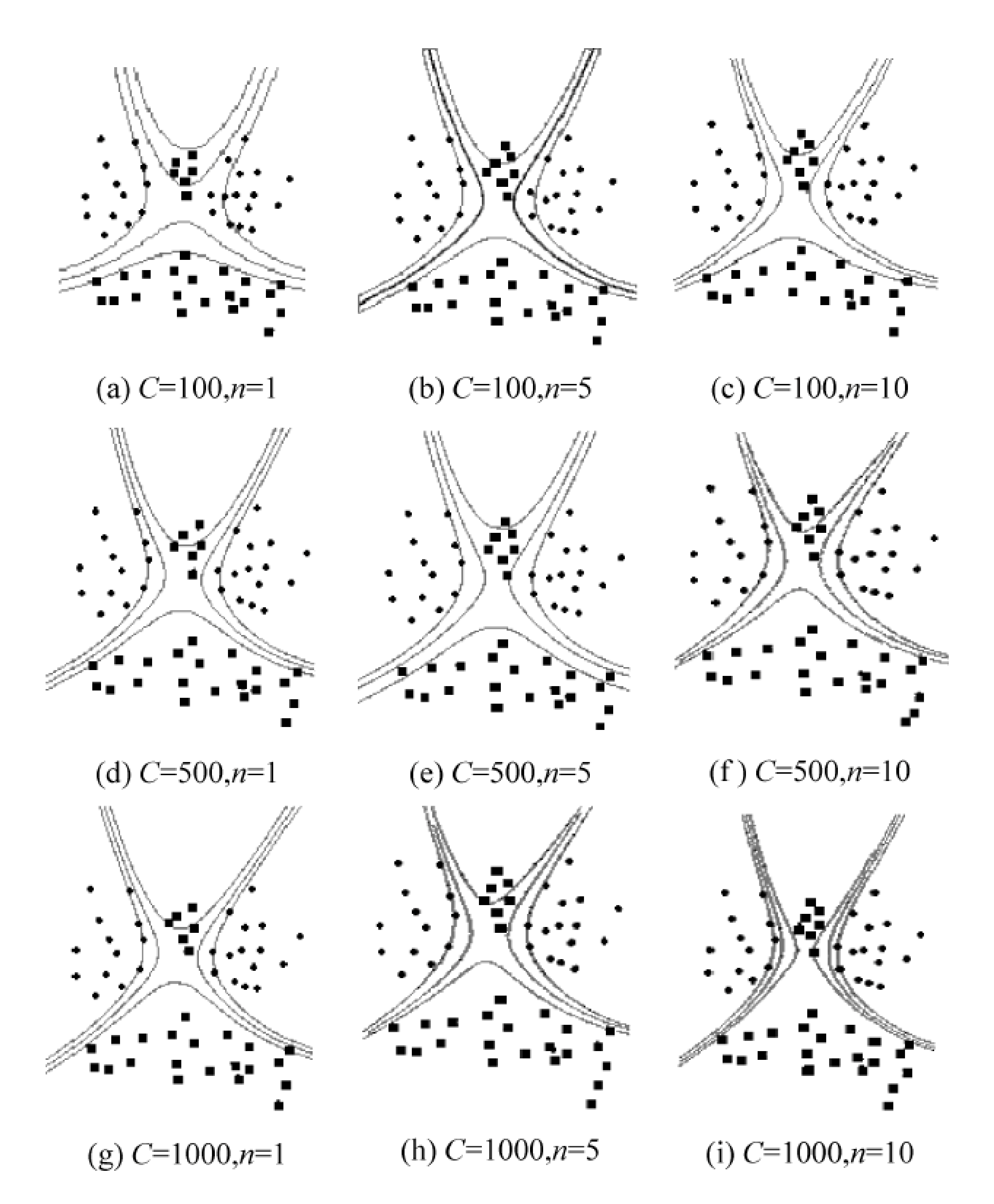


图 4-5　随 n 和 C 的取值不同 SLMBSVM 的分类决策面变化图

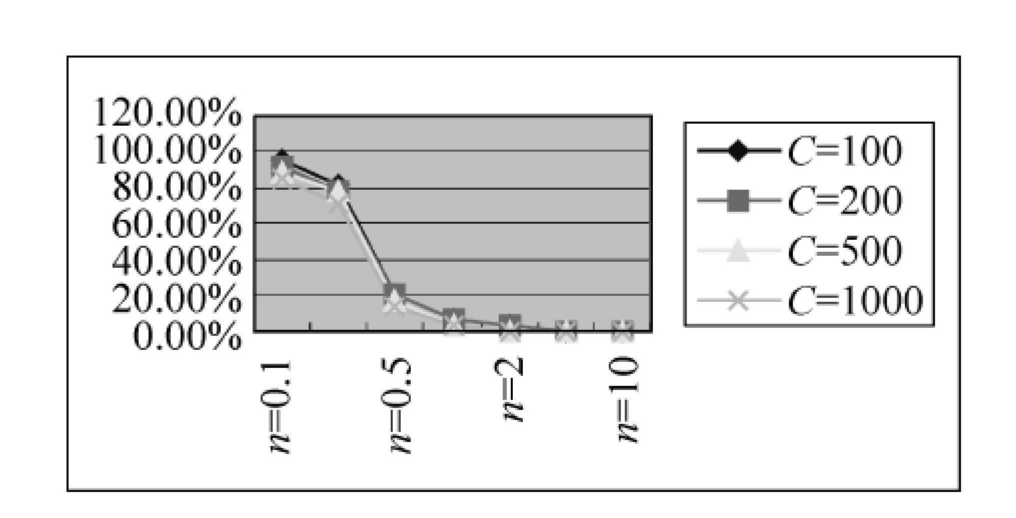


图 4-6　泛化 Pos_Error_Rate 结果

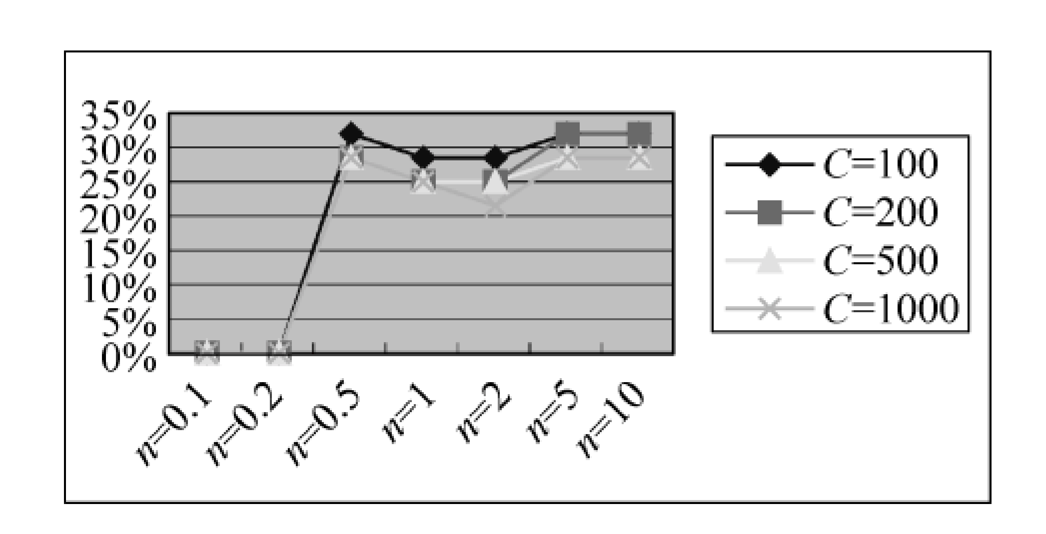


图 4-7　泛化 Neg_Error_Rate 结果

表 4-1　SLMBSVM 算法的泛化错误率和整体泛化损失

	n=0.1	n=0.2	n=0.5	n=1	n=2	n=5	n=10
C=100	1.6 96.4 0 0.0482 32(57.1%)	2.0 82.1 0 0.0821 34(60.7%)	2.6 21.4 32.1 0.214 37(66.1%)	2.4 7.1 28.6 0.1785 31(55.4%)	2.5 3.6 28.6 0.179 26(46.4%)	1.7 0 32.2 0.1610 24(42.9%)	1.6 0 32.2 0.1610 23(41.1%)
C=200	1.6 92.9 0 0.04645 33(58.9%)	2.0 78.6 0 0.0786 35(62.5%)	2.5 21.4 28.6 0.1965 37(66.1%)	2.7 7.1 25 0.1605 30(53.6%)	2.4 3.6 25 0.1610 26(46.4%)	1.9 0 32.2 0.1610 24(42.9%)	1.9 0 32.2 0.1610 22(39.3%)
C=500	1.8 89.3 0 0.04465 32(57.1%)	1.9 78.6 0 0.0786 34(60.7%)	2.7 17.9 28.6 0.18775 37(66.1%)	2.3 3.6 25 0.143 29(51.8%)	2.3 0 25 0.1250 25(44.6%)	1.8 0 28.6 0.1430 23(41.1%)	1.7 0 28.6 0.1430 22(39.3%)
C=1000	1.7 85.7 0 0.04285 32(57.1%)	2.0 71.4 0 0.0714 34(60.7%)	2.4 14.3 28.6 0.17875 35(62.5%)	2.6 3.6 25 0.1430 29(51.8%)	2.2 0 21.4 0.1070 26(46.4%)	1.7 0 28.6 0.1430 22(39.3%)	1.6 0 28.6 0.1430 21(37.5%)

注：表格中的实验数据依次为执行时间(s)、正样本识别率(%)、负样本识别率(%)、整体错误率和支持向量个数的比率(%)。

4.1.8　SLMBSVM-kNN 算法

SVM 对于线性不可分的问题，通过引入非线性映射将输入向量映射到高维空间，SVM 能在高维空间中给出最佳分类超平面。就分类精度而言，SVM 是目前分类方法中较好的一个。李蓉、叶世伟、史忠植等将 SVM 和 kNN 相结合，提出的 SVM-kNN 分类器，更进一步提高了网页分类的精度。其主要思想是：对于待识别样本 x，计算 x 与两类支持向量代表点的距离差，如果距离差大于给定的阈值，即 x 离分界面较远，用 SVM 分类一般都可以。当距离差小于给定的阈值，即 x 离分界面较近，则采用 kNN 对测试样本分类。这种方法的困难之处在于阈值的确定上。因为阈值与特定的问题有关，且其绝对值变化较大。因此在本节中提出了一种基于相对参数 n，两类损失比(百分数)来构建基于结构损失最小化的支持向量机(SLMBSVMs)的数学模型，随后再将其和 kNN 结合提出一种文本分类的算法。

借助 SLMBSVM 的特性提出了一种用于两类文本(文档)分类的算法，该方法的特点是构建两个损失最小化的 SVM，分别对训练样本进行判定，当 SVM1 判定属于正类时，算法直接输出正类，当 SVM2 判定属于负类时，算法直接输出负类，如果 SVM1 判定属于负类且 SVM2 判定属于正类，这时应该交给 kNN 来判定。

SLMBSVM-kNN 分类器算法如下：

输入：训练样本集 D，参数 $n \geqslant 1$，$C \geqslant 1$，测试集 T，kNN 的个数 k。

输出：分好类的测试集。

处理：

(1) 给定参数 $n \geq 1$，$C \geq 1$，针对于训练样本集 D 利用 SLMBSVMs 算法求出相应的支持向量集(SV1)和它的系数以及常数。

(2) $C=C \times n$，$n=1/n$，针对于训练样本集 D 再利用 SLMBSVMs 算法求出相应的支持向量集(SV2)和它的系数以及常数。

(3) 如果 $T \neq \Phi$，取 $x \in T$；如果 $T=\Phi$，停止。

(4) 利用支持向量集(SV1)对测试样本进行分类。如果分为正类，输出正类。如果分为负类，利用支持向量集(SV2)对测试样本进行分类。如果分为负类，则输出负类。如果分为正类，跳转到(5)。

(5) kNN 算法判断该样本的类别，并输出。

(6) $T=T-\{x\}$，跳转到(3)。

4.1.9　m-SLMBSVMs 与 kNN 相结合的多类文本分类算法

SLMBSVM-kNN 算法是用来解决两类分类问题的。解决多类的分类问题可以将多类问题转化为两类分类问题来解决。目前将多类问题转化为两类分类问题主要有两种方法，一种是"1-vs-1"方法，另一种是"1-vs-rest"方法。在文本分类中，采用 1-vs-rest 策略来将多类问题转化成 SVM 可以解决的两类问题。训练正例是该类所包含的全部网页；而反例是在训练集中不属于该类的所有其他类的网页。但在其多类网页分类问题中，选择基于损失 SVM 分类器的顺序对整体的训练精度有很大影响，因此在选择时按照以下算法来选择分类器顺序，即在选择分类器时采用剩余分类器中最小错误率的分类器的方法，该算法经实验证明简便有效。

m-SLMBSVM-kNN 多类网页分类的具体算法如下：

输入：训练样本集 D，参数 $n \geq 1$，$C \geq 1$，测试集 T，kNN 的个数 k，$M=\{M1, M2, \cdots, Mm\}$，为类别个数，M_i 为类别号。

输出：分好类的测试集。

处理：

(1) 给定参数 $n \geq 1$，$C \geq 1$，针对训练样本集 D 利用 SLMBSVMs 算法求出每个 1-vs-rest SVM 分类器相应的支持向量集(SV1)和它的系数以及常数。

(2) $C=C \times n$，$n=1/n$，针对于训练样本集 D 再利用 SLMBSVMs 算法求出每个 1-rest SVM 分类器相应的支持向量集(SV2)和它的系数以及常数。

(3) 如果 $T \neq \Phi$ 且 $|M|>1$，取 $x \in T$；如果 $T \neq \Phi$ 且 $|M|=1$，输出剩余 x 属于剩余类 M，停止；如果 $T=\Phi$，停止。

(4) 找到剩余的分类边界最大的支持矢量 SV1：

$$i = \underset{i, i \in M}{\arg\max} \ |\ margin(\mathrm{SV1}_i)\ |$$

(5) 对每一步 $x \in T$：

① 利用支持向量集($\mathrm{SV1}_i$)对测试样本进行分类：如果分为正类，输出 x 属于类 i，$T=T-\{x\}$。如果分为负类，利用支持向量集($\mathrm{SV2}_i$)对测试样本进行分类：如果分为负类，则不作输出，说明 x 不属于类 i，跳转到(6)；如果分为正类，跳转到②。

② kNN 算法判断该样本的类别，如果属于类 i，则输出 x 属于类 i，$T=T-\{x\}$。

(6) $M=M-\{M_i\}$ 跳转到(3)。

4.2 基于查询词关联的关键词集发布研究

本章首先介绍了多关键字检索所涉及的相关技术和研究现状，然后针对于目前多关键字检索系统的不足提出了基于检索关键词关联的关键词集发布的多关键词检索系统。

4.2.1 引言

P2P 文件共享的应用需求直接引发了 P2P 技术热潮，而文件共享和快速信息检索的关键是索引机制的建立。本节将阐述了如何高效建立关键字的索引以实现对等网络的多关键字检索。

许多 P2P 系统使用分布式倒排索引技术来帮助快速查找一个给定词出现的文档，用来索引数据对象的结点是基于给定的哈希算法来精确决定的。分布式倒排索引在多关键字检索中带宽消耗较大，为了减少请求开销，KSS[18] 使用关键字集来划分索引。但是，由于关键字集合远远大于个体词汇，一个 KSS 索引比一个标准索引大许多倍。即使 KSS 利用距离窗口技术[18] 在文档集中进行全文查找所需要的插入开销和存储开销也是无法接受的。

KSS 即关键字集搜索系统，它使用分布式倒排索引技术。在传统的分布式索引系统中，最大的一个挑战就是找到一个恰当的结构来分割索引。用关键字来分割是个不错的选择，但要求每个关键字的索引入口列表能够被检索到，这样这些列表之间才可以做交集。尽管在路由时不需要经过太多的结点，但是每经过一个结点都要发送大量的数据。而在 KSS 中，索引是由关键字集来分割的。KSS 建立一个倒排索引，将每个关键字集映射到一个文档列表上，当然这个文档列表必定包含关键字集中的字。当用户搜索时，查询关键字就被分割成一个个关键字集，然后可以从网络中得到各个集合的文档列表，再将它们做交集就输出了最终匹配的文档列表。

4.2.2 相关工作

在过去几年中，对等网络的查询和分布式索引已经被广泛研究。最近几年开发的许多 P2P 系统和算法都支持对等网络的查询，先来回顾一下相关工作。

在 Napster 中的查询索引是集中式的，文件的存储和服务是分布式的。当一个结点加入时，它将给 Napster 服务器发送一个文件列表，其中包含了自己共享的所有文件。如果有用户正想在 Napster 上搜索音乐文件，并发送查询消息，那么 Napster 服务器就搜索它本地的索引，一旦找到，用户就可以从对应的结点上直接下载想要的那个文件。从文件共享的角度来看，Napster 是一个 P2P 系统，然而，从索引方面看，Napster 是一个集中式的系统，类似于目录查询。

Gnutella[19] 也是一个 P2P 文件共享系统。它的搜索原理是将查询消息使用泛洪(Flooding)方式向所有邻居结点广播，并使用一个的网络生存时间(TTL)来进行范围限制以避免造成网络拥塞。也就是说，它在网络中的搜索方式是广度优先(BFS)搜索，直至找到需要的文件。发出查询的结点起初不知道哪个结点上可能会共享着它想要的那个文件，所

以它就只能将查询广播给它所有的邻居。虽然它很简单且能够发现存在于临近领域中的大量对象,但这种方法不能解决查询时需要访问大量结点所产生的巨大开销。Gnutella 网络采用的是非集中式的结构,不建立中心目录服务器,文件的位置和文件查询、下载也都是纯分散形式的。

对于 Gnutella,由于其广度优先(BFS)搜索消息流量过大,参考文献[20]中提到的迭代加深(ID)就可以将每一个搜索所消耗的带宽降到最低,同时保证质量,在比较合理的时间内得到理想的结果。ID 不必浪费太多的带宽,直到网络中 N 层,它只要结点将查询广播至 $N-k$ 层即可。假若没有找到目标,那么将深度继续迭代直至找到。ID 有助于节省带宽消耗,利用 BFS 搜索方式,Gnutella 网络上的大多数搜索都很有可能在一个很小的深度值的时候找到结果。

Gnutella2[20]中当一个超级结点接受到一个来自叶子结点的请求时,它将请求传递到与它相关的叶子结点及与它相邻的超级结点。这些超级结点在当地处理请求并传递给与它们相关的叶子结点。相邻的超级结点有规则地交换当地存储库表格以过滤出在他们之间的不必要的流量。Modified-BFS[21]、Intelligent-BFS[21]和 APS[22]是 Flooding 机制的一种变化,终端随机或智能的选择一定比率的邻居来传递请求。与以前的方法相比,这些算法确实减少了平均消息产生,其准确度和效率仍然是一个问题。如果反馈时间是一个很关键的因素,那么就可以使用定向 BFS。这是一个广播策略,在邻居列表中选择合适的几个邻居结点来发送查询消息,这样就避免了搜索所有的邻居,这种方法又称 DBFS。在选择邻居的时候,原则是参照邻居在过去搜索中的性能表现。DBFS 是纯 BFS 和纯深度优先搜索的折中策略。

本地索引(Local Index)[23]的原理是采用结点缓存机制预先存储距离本地结点 r 跳以内的多个结点信息,并允许 P2P 网络中的一个结点来代表这些结点来回答某一条查询,这样就不需要把查询消息往前继续传递了。这个系统使得 D 跳以内的文档在 $D-r$ 的 TTL 内被搜索到。然而该系统同样有建立和查询索引的巨大流量开销。在建立索引时,一个结点要涉及 r 跳数以内的所有结点;在搜索时,$D-r$ 个跳数以内的结点仍然要被查询到。查询以类似于 BFS 的方式进行,但是只有那些在某个深度接近请求者的结点处理请求。由于每个相连的结点索引许多终端,这种方法的准确度和命中率非常高。另一方面,消息产生与 Flooding 模式是可比的,虽然并非所有的结点处理请求,但其处理时间还是相当小的。

上面所描述的都是基于非结构化 P2P 系统的查询。接下来我们讨论结构化 P2P 网络中的查询模式。像 Tapestry[24]、Pastry[25]、Chord[26]和 CAN[27]这些协议模型使用不同的技术传播(键值对)。虽然这些分布的哈希结构能够被用来创建倒排索引,但它们并不是非常有效的。

1. 分布式倒排索引

所谓的倒排索引,就是一个数据结构,它将单词映射到文件,帮助快速地查找到含有给定单词的文档。建立并更新一个倒排索引就是一个预计算的过程,把一个文档中可能要被搜索的单词提取出来。一旦单词被提取,一个连带的列表就生成了,它指示了每一个单词对应的文件的指针。当然必要的话,还可能包含在那个文件中出现这个单词的位置。

倒排索引可以在本地建立。搜索引擎(如 Google)就是遍历网络,在本地建立索引,然

后就可以用它来快速找到查询的结果；像 Napster 的 P2P 文件共享系统也是这样，使用一个服务器本地的索引，来搜索文件，当一个用户在 Napster 服务器上注册并共享一个文件后，这个索引就会建立。

而在分布式索引系统中，索引并不是整个都存储在一个服务器上，而是将索引分散地放在多个网络结点上。分布式索引中最大的挑战便在于，怎样找到一个正确的结构去分割结点上的索引。有一个简单的解决方法是用文档中的关键字来分割索引，针对某一给定的关键字，将所有的索引入口存储在一个特定的结点上。在这样一个结构中，要处理一个查询时，系统只须获得查询中每个单词的索引入口，然后将得到的结果集合起来就可以了。例如，用户的搜索字是"对等网络"，那么系统会下载所有"对等"和"网络"的索引入口，然后将列表做交集即可。但是有成百上千的文件含有"对等"，又有成百上千的文件含有"网络"，所以，对于搜索字中每一个关键字，系统都需要传送大量的索引入口，甚至有可能是兆级的字节大小。图 4-8 就是一个 P2P 网络中的倒排索引模式，它分散存储在多个结点上。

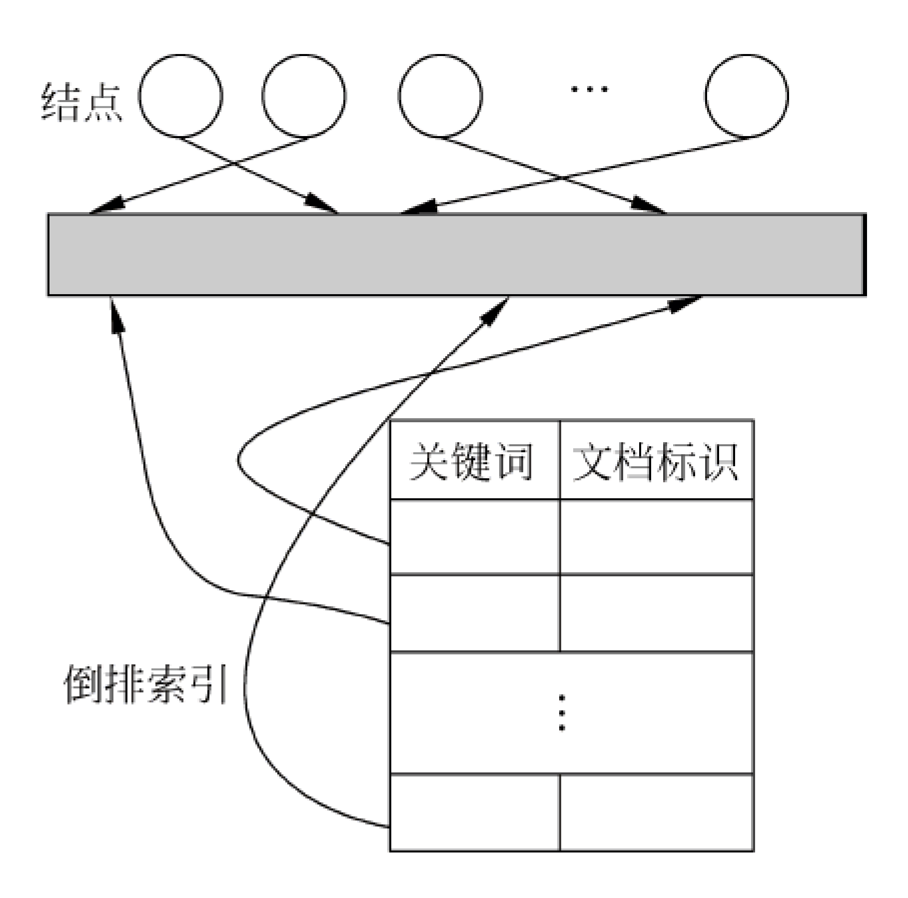


图 4-8　P2P 网络中的倒排索引

从多个结点上获取索引入口，并将结果做交集，这需要一定的开销，而降低这个开销的一个标准优化方案，就是要将较短的列表传送给列表较长的结点。但是，由于"对等"的列表和"网络"的列表都比较长，所以看来，这样的优化对于 P2P 搜索效率并没有什么太大的帮助。

2. 关键字集搜索系统（KSS）

Omprakash D Gnawali[18]提出了一种基于关键字集的查询系统（KSS），其是一个使用倒排索引的 P2P 搜索系统。它使用一种分布式倒排索引，即通过一个关键字集合划分网络中结点的索引。KSS 建立了一个分布式倒排索引，将每一个关键字集合映射到一个包含关键字-集中单词的所有文档的列表。关键字集倒排索引是这样输出的：假如在一个文档 doc1 中含有关键字"对等"、"网络"和"检索"，并且字集大小是 2，那么索引入口就是＜"对等网络"，doc1＞，＜"对等检索"，doc1＞，以及＜"网络检索"，doc1＞，对于字集大小是 3 的情况，入口就是＜"对等网络检索"，doc1＞。所有对于某个关键字集的索引入口都存储在某一个特定的结点上，而这个结点则是通过对关键字的集合进行哈希挑选出来的。

众所周知，用户是通过关键字来搜索文件的。我们将用户查询中的关键字分割成几个

关键字集合，于是就能从网络中获取每个集合对应的文件列表。比如说，用户输入“对等”，“网络”，“检索”这样三个关键字，系统就获取关于“对等网络”、“网络检索”和“对等检索”的索引入口，一旦所有的列表到达，在所有列表中都出现过的文件就是与查询相匹配的了。假如我们得到的关于“对等网络”的索引入口有＜“对等网络”，doc1＞，＜“对等网络”，doc2＞，＜“对等网络”，doc3＞，“网络检索”的索引入口有＜“网络检索”，doc1＞，“对等检索”的索引入口有＜“对等检索”，doc1＞，＜“对等检索”，doc2＞，那么与那个查询相匹配的文档就是 doc1 了。

对于 KSS 的索引，我们可以想象，它会比一个普通的倒排索引大得多，因为关键字集的个数明显大于单个关键字的个数，关键字的单个个数越多，这个差距就越大。但是 KSS 的一个显著的优势在于，在执行一个多关键字查询时，它的通信开销是相当低的。在一个典型的 KSS 设置中，双词查询不需要大量的通信，这是因为它们全部被负责该关键字集的结点中索引。多于两个词的请求需要增加传输文档列表的开销，但是由于那些包含两个单词集的文档数目比包含单个单词的文档数目要小的多，因此通信开销也较小。虽然对于目标应用的请求开销被减少，但是插入开销却随着那些被用来产生索引入口关键字的关键字-集的大小呈指数级增长。也就是，相比于传统单个单词的发布，KSS 导致了更多的插入开销。

KSS 是使用分布式倒排索引，要索引一个文档，KSS 将这个文档中所有关键字都提取出来，然后对关键字的列表做排序，删除其中有重复单词的，这样的一个无重复字的有序列表，称为全字集。然后，KSS 从“全字集”中输出索引入口。假如关键字集的大小是 k，那么 KSS 就为全字集中所有 k 个字的组合产生索引入口，此时若文档有 n 个字，那么所产生的索引入口就有 $C(n,k)=\frac{n!}{(n-k)!\,k!}$ 个，其中索引的 key 就是由入口中的关键字集的哈希形成的。然后，就可以用 Chord 将每个索引入口的 key 映射到 P2P 网络中的结点上。

尽管 KSS 的用户可以指定很少的 Metadata 域的单词来发布文件，以减少插入开销，但这种做法不能满足全文检索。

3. Bloom Filter 技术

分布式倒排索引技术将每个关键字映射到存储该关键字的相应结点，在执行多关键字查找时，牵涉到传送文档列表并计算交集。显然这种检索机制需要在网络上传输大量的中间文档列表数据，产生大量的网络带宽开销，当前有许多研究集中在如何减少带宽开销以及提高检索效率。在减少带宽开销方面，参考文献[28]中采用了 Bloom Filters[29] 机制，如图 4-9 所示。例如，进行组合查询 $A\cap B$，设关键字 A 映射到结点 A，关键字 B 映射到结点 B。查询请求首先发往结点 A 检索包含关键字 A 的文档集合。在结点 A 上检索到包含关键字 A 的文档集合，然后计算该集合的 Bloom Filter 并将结果 $F(A)$ 发往结点 B，结点 B 检索本地得到包含关键字 B 的文档列表，并利用 $F(A)$ 计算出满足 Bloom Filter 测试的 B 文档集，记为 $F(A)\cap B$。该结果集再传递回给结点 A，并计算和 A 文档集的交集，将结果发送回客户端。

另外，结点 B 也可以直接将 $F(A)\cap B$ 发送给查询客户端，但这会包含一些“正向错误”。

在多关键字查询中，Bloom Filter 是常用的减少传输占用带宽的方法。此外，不同结构的结果缓存技术[30,31]也常用来避免重复查询和减轻通信流量。

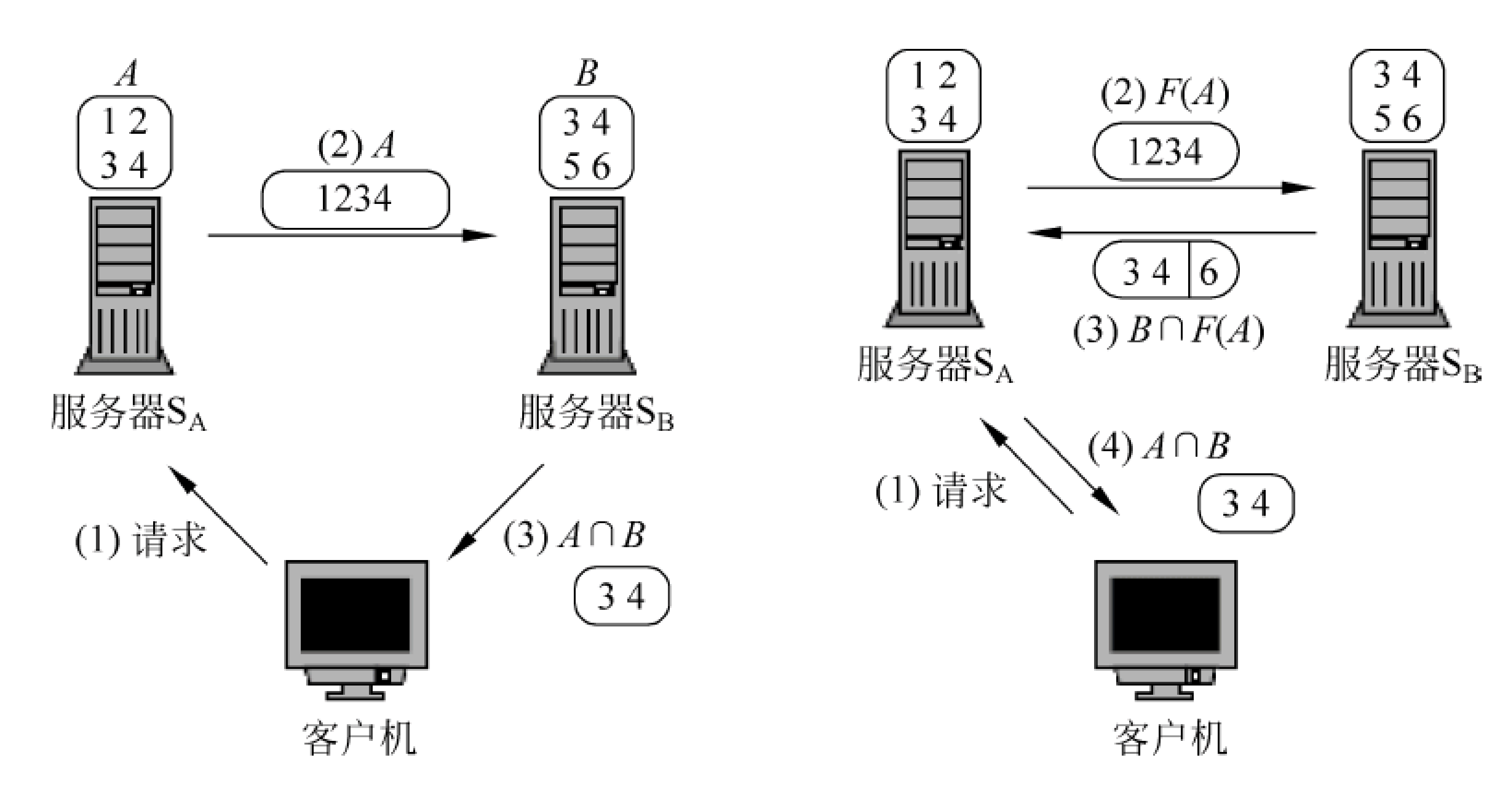


图 4-9　使用 Bloom Filter 计算组合查询 $A\cap B$[28]

4.2.3　KRBKSS 系统模型

我们发现在典型的 KSS 中许多映射到网络结点的关键字对并不或很少在用户的真正请求中被使用,也就是说用户对多关键字的组合查询并不是等同概率均匀分布的,而是有相关性分布的,其原因是在 KSS 中请求关键字间的联系并没有被考虑到。鉴于此,希望能借助用户以前提交的查询关键字集来挖掘出查询关键字之间的内在联系,进而指引结点的发布。我们建立用户查询关键字关系模型来提高 KSS 的插入和存储效率,提高 KSS 系统的性能。

构造 KRBKSS(Keywords Relationship Based Keywords Set System)的基本过程如图 4-10 所示。构造 KRBKSS 系统包括两个阶段:

(1) 使用关键字关系发现算法(Keywords Relationship Discovery Algorithm, KWRDA)来发现来自于请求日志的请求关键字之间的联系,请求日志可以从 FTP,WWW 搜索网站的日志中获得。

(2) 在 KRBKSS 中,只映射这些通过第一步中 KWRDA 输出的边,而不是在关键字集合查询系统中所有的关键字对(集)。

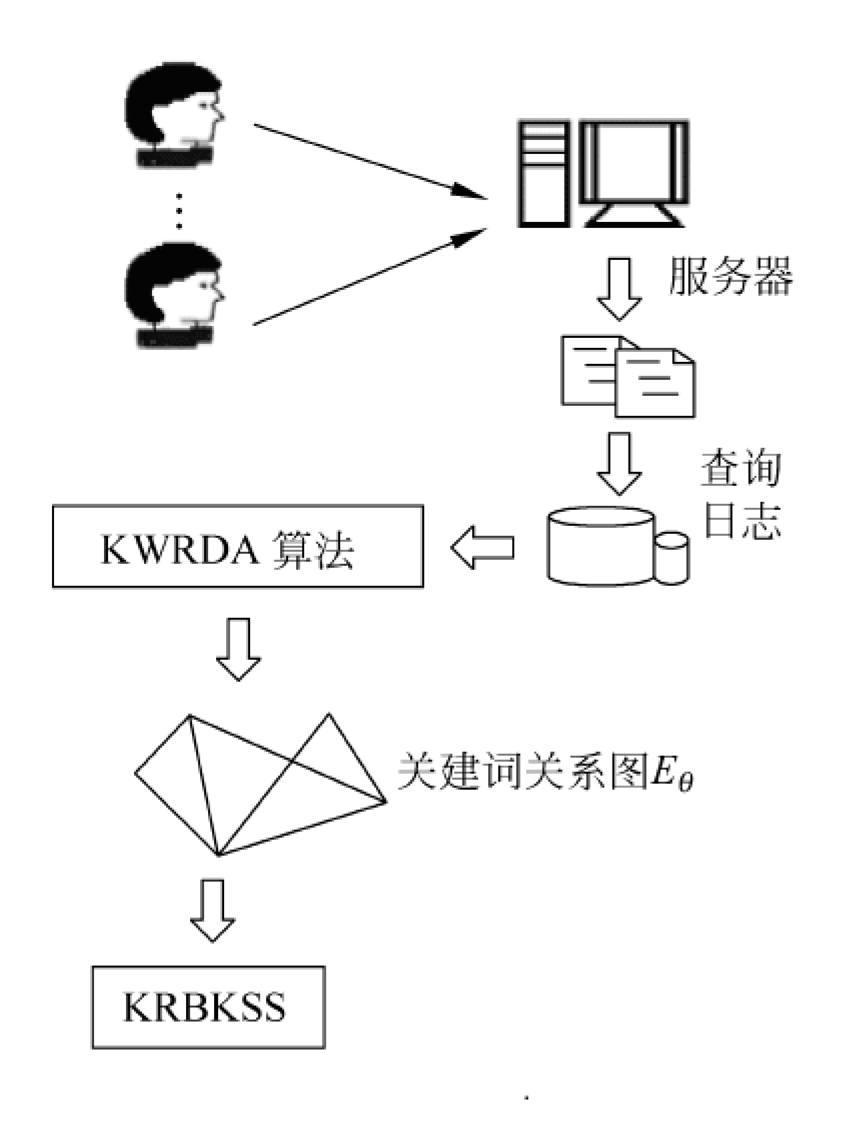


图 4-10　构建 KRBKSS 的过程

1. KWRDA 算法描述

用户请求的关键词之间的联系对于文档的分布式倒排索引是非常有用的,因此提出了 KWRDA 算法。KWRDA 算法将服务器的访问日志作为输入,关键词间关系(基于查询相关性)为输出。

进行关键词关系发现有两种方法:一种是利用关键词间查询的支持度来进行构建,也即概率构建算法;另一种是利用数据挖掘中关联挖掘的思想来进行多关键字关系发现,也即关联挖掘算法。

1) 概率构建算法

概率构建算法将服务器的访问日志作为输入,并将

其映射成为一张表现关键词之间联系的图。该算法有三个主要步骤：

（1）根据请求日志构造一张有向图 $G(A,E)$。

（2）根据一个给定的连通性阈值 θ 将图修改为$G(A,E)|_{\theta}$。

（3）输出 E_{θ}。

下面依次讨论每个步骤。

（1）根据请求日志构造一张有向图 $G(A,E)$。

定义 4-1　用户 U_i 在时刻 t 所提交的查询中的查询词集用$\{k_0,k_1,\cdots,k_{p_t}\}_{i,t}$来表示，其中 $p+1$ 为该次查询的查询词个数。

定义 4-2　一次查询：设定 30 分钟内用户提交的查询为一次查询。

定义 4-3　查询周期 T_j：通常设定一段时间为一个查询周期，用来反映用户查询行为的阶段性，一般为一个月。

定义 4-4　用户 U_i 在查询周期 T_j 内的查询词集：用户 U_i 在某个查询周期 T_j 内每次查询所提交的查询词的集合，用 $\bigcup_{t\in T_j}\{k_0,k_1,\cdots,k_{p_t}\}_{i,t}$来表示。

定义 4-5　所有用户在查询周期 T_j 内的查询词集：所有用户在某个查询周期 T_j 内所提交的查询词全集，用$\bigcup_{\forall i}\bigcup_{t\in T_j}\{k_0,k_1,\cdots,k_{p_t}\}_{i,t}$来表示。

定义 4-6　查询单项：查询词全集 $\bigcup_{\forall U_i}\bigcup_{t\in T_j}\{k_0,k_1,\cdots,k_{p_t}\}_{i,t}$中每一个查询关键词称为一个查询单项。

定义 4-7　用户 U_i 在查询周期 T_j 内查询查询单项 X 的查询次数，用 $\text{query_num}_{U_i,T_j}(X)$来表示。

定义 4-8　所有用户在查询周期 T_j 内查询查询单项 X 的查询次数，用 $\text{query_num}_{U,T_j}(X)$来表示。

$$\text{query_num}_{U,T_j}(X) = \sum_{\forall i}\text{query_num}_{U_i,T_j}(X)$$

定义 4-9　查询关键词 A_1 对查询关键词 A_2 的支持度。

$$\text{SUP}_{T_j}(E_{A_1\to A_2}) = \frac{\text{query_num}_{U,T_j}(A_1\cap A_2)}{\text{query_num}_{U,T_j}(A_1)}$$

类似地，我们可以知道查询关键词 A_2 对查询关键词 A_1 的支持度。

$$\text{SUP}_{T_j}(E_{A_2\to A_1}) = \frac{\text{query_num}_{U,T_j}(A_1\cap A_2)}{\text{query_num}_{U,T_j}(A_2)}$$

在图 $G(A,E)$中，顶点的集合 A 对应于在用户请求中使用的查询单项。边的集合 E 对应于在用户请求中观察到的查询单项的共发性。$E=\{e|\text{SUP}(e)>0\}$。由于 $G(A,E)$是一张有向图，$E_{A_1\to A_2}$ 及 $E_{A_2\to A_1}$ 应该被互相区分。其中 A_1 和 A_2 是集合 A 中的顶点。例如，如果一个请求包括查询术语“P2P”和“search”，则对应边的查询次数会相应的加 1。有向边(p2p→search)上的权值表示的是关键词 P2P 对关键词 search 的支持度。

（2）根据一个预先设定的连通性阈值 θ 将图修改为$G(A,E)|_{\theta}$。该算法对于所有用户的所有查询涉及的边，如果图中该边不存在，则在关键字关系图中添加相应的边，如果该边存在，在相应边的查询次数上加 1；然后计算每条边起始结点关键词对终点关键词的支持度；最后检查每条边，看看是否大于给定阈值，如果小于给定阈值，则删除该边，剩下的图即为$G(A,E)|_{\theta}$。

由于图 G 的连通性一般是较高的，我们使用一个连通性阈值来减少图中边的数目。连通性阈值表示为现存边的最小权值。当这个阈值变大，图将变得稀疏，当阈值变小，图将变得密集。

在图 $G(A,E|_\theta)$ 中顶点集合 A 与图 $G(A,E)$ 中的顶点集合 A 相同，对应于用户请求中使用的查询单项。然而，边的集合 $E|_\theta$ 对应于在用户请求中观察到的查询术语共发性，即

$$E|_\theta = \{e \mid e \in E \text{ and } \mathrm{SUP}(e) \geqslant \theta\}$$

显然不同的连通性阈值 θ 会输出不同的 E_θ。连通性阈值 θ 越大，图越稀疏。

(3) 输出 E_θ。这一步输出对应于 E_θ 的顶点对，代表了查询关键字间的并发关系。

该方法的具体算法如下：

算法 4-1 基于词间支持度的关键词关联发现算法。

```
input:时间段 T_j 内的所有用户的所有次查询
output:关键词关系图
for all U_i
    for all query
        insert into graph(K1,K2)        // 任意两个关键词
        num(K1,K2) ++ ;
for each edge
    compute SUP_Tj (E_A1 ->A2 );
for each edge in G(A,E)
    if SUP_Tj (E_A1 ->A2 ) <  threadhold  delete(e)
    return G(A,E)|_θ;
```

2) 关联挖掘算法

概率构建算法的缺点在于：一方面 θ 值的确定有人为性，另一方面只能进行两个关键字间关系的构建，而不能进行多关键字间关系的构建。因此，我们提出利用数据挖掘中关联挖掘的方法来进行频繁项挖掘以发现查询相关的多个关键词。同样为了和算法 4-1 概率算法结合使用，可以由概率算法生成双关键字间联系，由关联挖掘算法在其基础上生成多关键词间联系，并将两者合成来指导发布。借鉴 Apriori 关联算法，我们的具体算法见算法 4-2。首先，对一些基本定义做一下说明：

item：i_j 即查询单项。

itemset：$I=\{i_1,i_2,i_3,i_4,\cdots,i_m\}$ 即关键词的查询单项集合。

Ck：代表可能成为集合大小为 k(含有 k 个 items 的 itemset)的候选集项。

Lk：代表集合大小为 k(含有 k 个 items 的 itemset)的 frequent itemset，itemset 中的每个 item 皆满足至少在日志中出现的概率不小于支持度。

算法 4-2 使用频繁项集关联挖掘的关键词关系发现算法。

```
C2 = {frequent 2 - itemset};              //根据概率算法给出的两关键字之间的关系，形成 C2
for (k = 2; |Ck|≠0 and k<5; k++)          //仅发现 4 个以下关键词间的关系
begin
    对于 Ck 中的每一个集合，验证是否是频繁的;      //可以假定 C2 都为频繁的
    令 Lk 成为 Ck 中频繁集的汇集;
    生成候选关联关系 Ck+1;
end
```

2. 系统结构

KRBKSS 能够被嵌入到任意支持分布式哈希(DHash)表接口的 P2P 平台中，如 Chord、CAN 和 Tapestry。这一特点，类似于 KSS，我们将描述一个使用 Chord 系统的示例系统。KRBKSS 系统架构如图 4-11 所示。

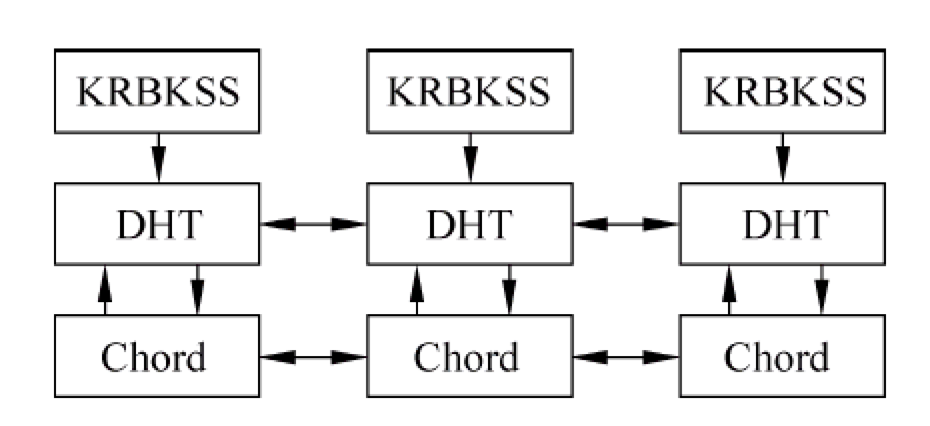


图 4-11　KRBKSS 系统架构

系统结构中的 Chord 层提供了一种操作的支持：给定一个键，它将键映射为一个结点。系统中的每个结点使用一致性哈希函数来维护一个称为邻接表的数据结构，通过这个数据结构请求被路由到越来越接近于目标结点或者其后继结点的地方。在一个 Chord 系统中每个查找操作所需的平均消息数目为 $O(\log n)$。

DHash 层在 Chord 系统上增加了一个分布式哈希表，Dhash 提供了一个简单的得到-放入 API，它使一个 P2P 应用能够在 P2P 网络的结点中放入一个数据项以及根据它们的 ID 从网络中得到数据。

KRBKSS 提供了客户端应用的两种操作。一种是插入文档操作，它通过关键词之间的联系从文档中抽取关键词，产生索引项目，并将它们存储在网络中。另一种是查找请求中没有联系的关键词集合的文档列表，并返回文档的交叉列表。

3. KRBKSS 发布和请求

KRBKSS 工作如下：当一个用户共享一个文件，KRBKSS 利用关键词之间的联系来产生文件的索引项目，通过哈希关键字-集为索引项目形成键，使用一致性哈希函数和 Chord 将键映射成网络中的结点。

KRBKSS 建立索引的算法如下：

```
word[0..n] = meta-data field
for (i = 0; i<n; i++)
  for (j = i+1; j<n; j++)
      if ((word[i], word[j])∈E|θ)
          set_add(keywords, concat(sort(word[i], word[j])));
for (i = 0; i<keywords.size; i++)
      push(index_entries, <hash(keywords[i]), documentID>);
```

举个例子，令 A、B、C、D 为标识且为 docID 文档中的单词。KSS 为 6 个组合(AB、AC、AD、BC、BD、CD)中的每一个建立索引项目。对一个大小为 2 的集合，$C(n,2)$给出了在 n 个唯一关键词的集合的叉积中唯一项目的数目。然而发现关键词对 AC、AD、BC、BD、CD 从不或很少成为用户的请求对。这样在 KRBKSS 中只有关键词对 AB 被映射成 P2P 网络中的结点。

在 KRBKSS 系统中多关键词请求算法工作如下：

输入：$Q=\{k_1,k_2,\cdots,k_n\}$。

输出：一个包含请求中所有关键词的文档列表。

处理：

(1) $Q'=Q$。

(2) 当 $Q'\neq\varnothing$ 或 $|Q'|\neq 1$ 时，依次执行(3)和(4)，否则跳转至(6)。

(3) 利用关键词关系 E_θ，从 Q' 中找到最接近的两个请求关键词 k_m 和 k_p。

(4) 找到为两个关键词存储索引项目的结点并取得列表。

(5) $Q'=Q'-\{k_m,k_p\}$，跳转至(2)。

(6) 将相关文档列表求交集，把结果输出。

4.2.4 模拟实验

本节采用仿真评估 KRBKSS 算法。为了分析 KRBKSS 算法对全文索引的开销和效率，我们开发了一个网络爬虫 crawler，使用网页 www.edu.cn 和 www.sohu.com 作为种子并递归地下载文本和 HTML 文件。通过网络爬虫下载了 42 238 个文本页(大小为 492MB)。为了比较 KRBKSS 算法和现存 KSS 算法，执行了 KSS 和标准倒排索引。在仿真实验中使 1800 个结点运行于带宽为 100Mbps 局域网中的两台个人电脑上，每台电脑有 1.7 GHz 处理器和 512MB RAM 并运行 Linux AS 3.0。

可以利用搜索网站请求日志来发现请求关键词之间的关系。

我们开发了一个可调整系统 iExtra，它完全运行于 Java。在预处理阶段，iExtra 分析 HTML 页并且去除无效字符。经过分析，中文项将通过最大匹配中文单词分段算法抽取和分离出来，得出的中文单词使用唯一 ASCII 字符串进行编码。同时选择一个终止符列表来过滤英文和中文的终止符。

使用 KRBKSS 算法模拟文档的插入和请求。接着在每个文本文件上运行 KRBKSS 算法以创建索引项目并将它们发布到相应的虚拟终端。通过使用不同连通性阈值 θ 时的插入开销、请求开销评估这些算法。

1. 插入开销

所谓的插入开销，就是当一个文档插入到系统中时，所需要传输的字节数。当一个用户请求系统共享出一个文件时，系统生成索引入口，插入到分布式索引中。对于每个文档，生成的索引入口越多，那么插入开销就越大。如果要插入一个 n 字的文档，用所有可能大小的关键字集的所有子集作为 key，生成索引入口，那么这个插入的开销将是 2^n。对于关键字对的结构，即 2 个字为一个关键字集大小的情况，开销则是 $C(n,2)$。这种退化的情况是将关键字集中的每个词的哈希作为 key，它的插入开销就很小。如果一个文档的大小是 n，关键字集的大小是 k，一个索引入口的大小为 s，那么与插入相关的整个的开销就是 $C(n,k)\times s$。

插入开销是指，当文档被插入到系统时所传递的字节数。当用户请求 KRBKSS 系统以共享文件时，系统产生被插入到分布式索引中的索引项目。不同于 KSS，在 KSS 中如果我们为一个有 n 个关键词的文档产生索引项目，对典型的关键字-对模式所需的开销注定是 $C(n,2)$，KRBKSS 只产生很小的索引项目，其结果所需的插入开销是很小的。

图 4-12 显示了索引项目产生数目对应于在一个使用标准倒排索引模式文档中单词数目的曲线图，图中包括 $\theta=0$ 的 KRBKSS，$\theta=0.05$ 的 KBRKSS 和窗口大小为 10 的 KSS。

图 4-13 显示了使用在窗口大小为 5 的 KSS 和 $\theta=0$ 并使用标准倒排索引模式的 KRBKSS 时每个文档(页)被插入到系统所产生的索引项目数目分布。图 4-12 和图 4-13 说明 KRBKSS 的插入开销比 KSS 小得多,但标准倒排索引模式高一点。

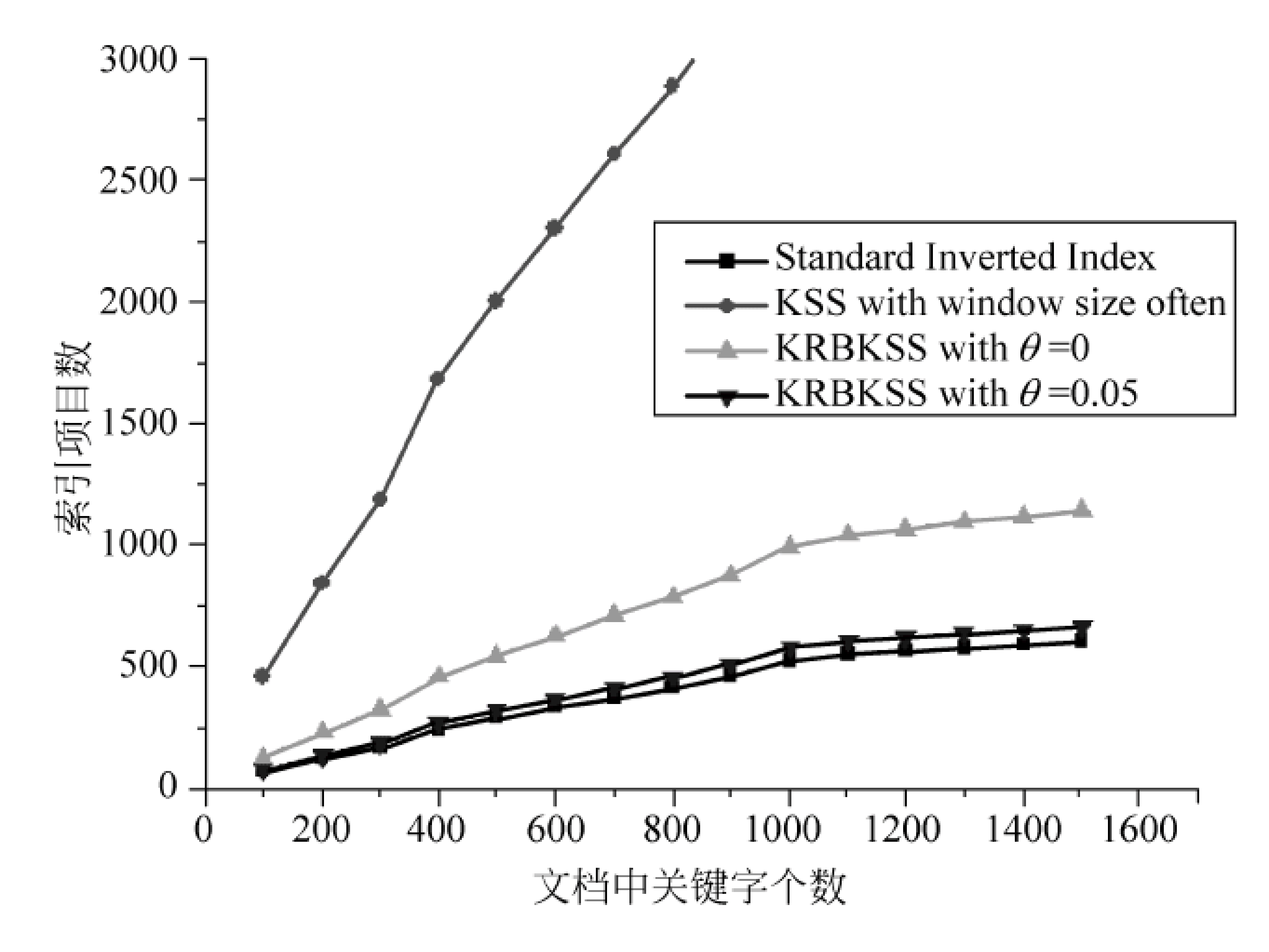


图 4-12　不同发布算法下发布量的变化关系

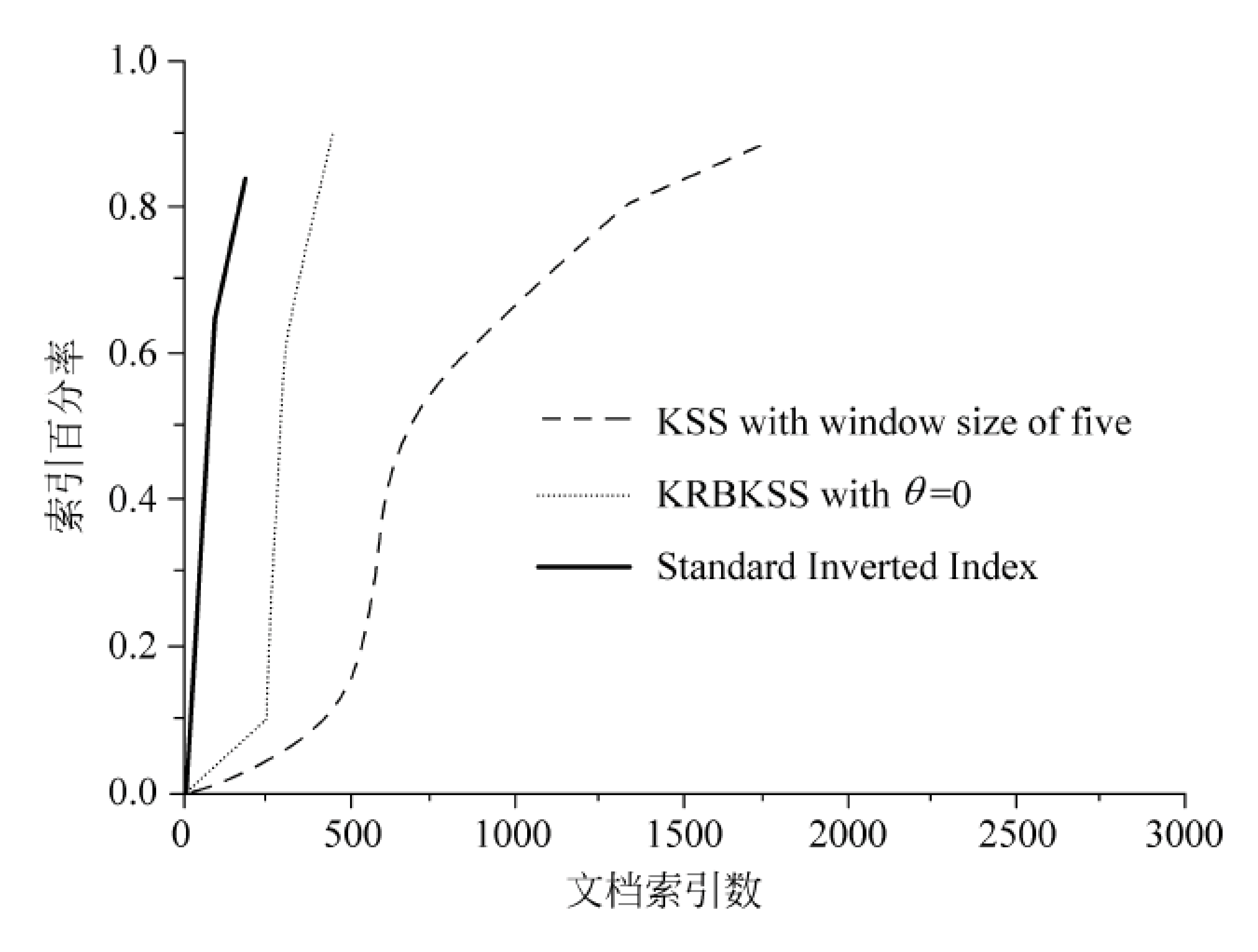


图 4-13　不同发布算法下每文档索引量和索引百分率的变化关系

2. 查询开销

查询开销,计算的就是当一个用户在系统中搜索某个文件时,需要传输的字节数。理想情况下,一个搜索系统只传输查询关键字的一个子集,以及用户感兴趣的匹配的结果。但是在一个现实的搜索系统中,只能尽可能地向这个理论的最小值靠拢。在查询中,传输搜索关键字并不是一个真正的开销,因为大多数查询只有很少的几个关键字。只有当这个查询要向很多台机器传输时,我们才会去考虑这个问题。真正的开销在于,在结果列表合成的过程中,要将中间的结果列表从一个结点传输到另一个结点。

查询开销是指当用户查找一个系统中的文档时字节的测量。众所周知,在系统中从一

个主机向另一个发送中间列表的开销是请求开销的主要部分。

图 4-14 所示为当使用 Bloom Filter 标准倒排索引，非 Bloom Filter 标准倒排索引，窗口大小为 5 的 KSS，$\theta=0$ 的 KRBKSS，$\theta=0.05$ 的 KBRKSS，对应于多种查询以 KB 传递的中间数据。图 4-14 说明 KRBKSS 的请求开销比 Bloom Filter 和非 Bloom Filter 的标准倒排索引低得多，但比 KSS 高一点。

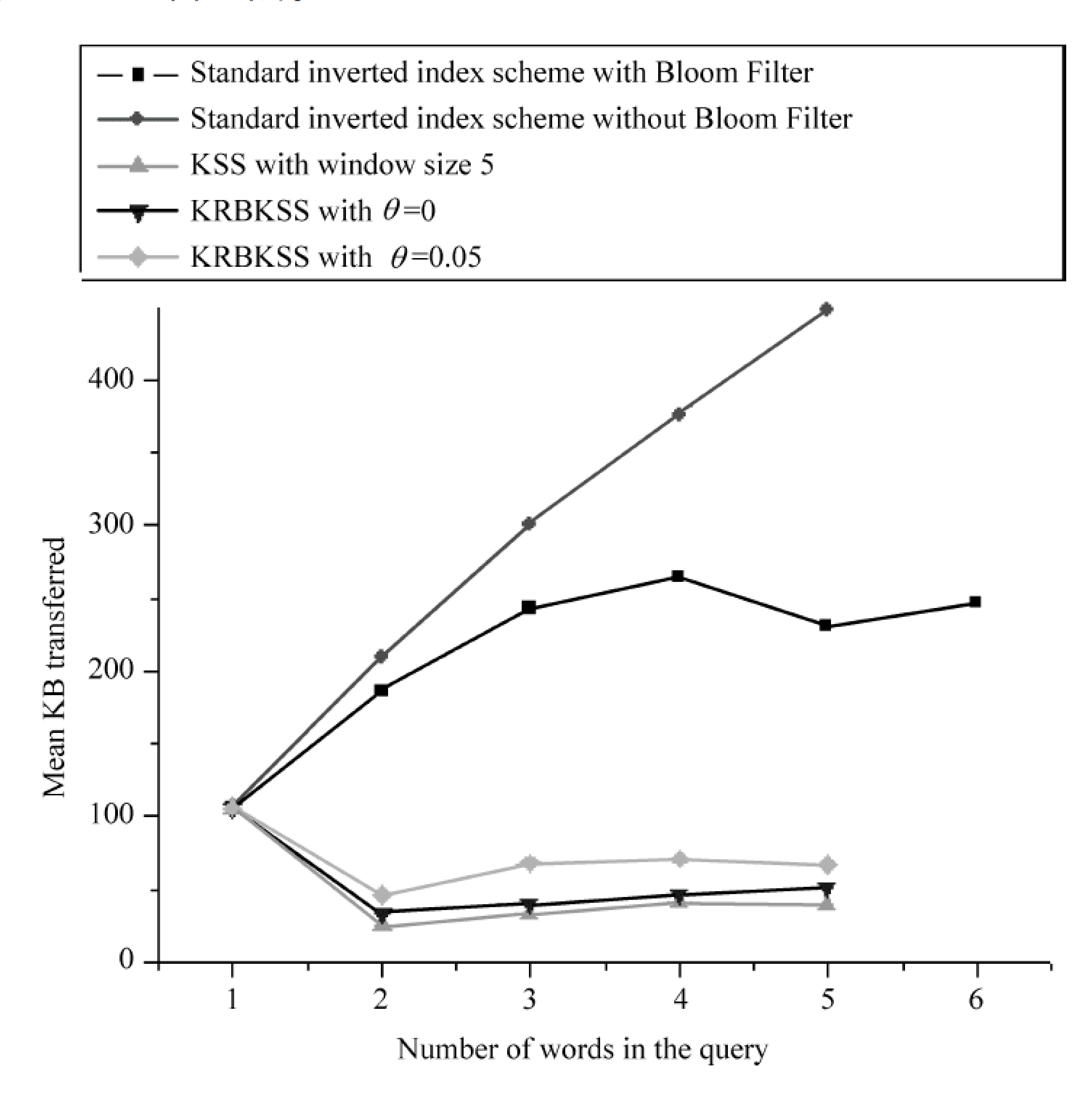


图 4-14　不同索引关键词个数下传输的字节量

4.3　本章小结

本章首先介绍基于分类器融合的对等网络文档分类算法研究，然后探讨了从用户请求日志中提炼的请求关键词之间的关系，以改进 KSS 系统的性能。其主要思想是：首先，利用 KWRDA 算法来发现来源于请求日志的请求关键词之间的关系，请求日志可以从 FTP 或 WWW 搜索网站获得。接着，在 KRBKSS 的发布中，仅映射这些 KWRDA 中的关联的关键词集合，而不是 KSS 中所有的关键词对，这样相比于 KSS 会极大地降低插入开销和存储开销。实验结果说明 KRBKSS 在插入开销和存储开销方面比 KSS 索引更有效，并且在请求的通信开销方面比标准倒排索引小。

参考文献

[1]　Fuhr N，Hartmann S，Knorz G，Lustig G，Schwantner M，Tzeras K. AIR/X—a rule-based multistage indexing system for large subject fields. In Proceedings of RIAO-91，3rd International Conference

"Recherche d'Information Assistee par Ordinateur". Barcelona, Spain, 1991: 606-623.

[2] Yang Y, Pedersen J O. A comparative study on feature selection in text categorization. in Proceedings of ICML-97, 14th International Conference on Machine Learning. Nashville, US: Morgan Kaufmann. 1997: 412-420.

[3] Yang Y, Liu X. A re-examination of text categorization methods. in Proceedings of SIGIR-99, 22nd ACM International Conference on Research and Development in Information Retrieval. 1999: 42-49.

[4] NG H T, Goh W B, Low K L. Feature selection, perceptron learning, and a usability case study for text categorization. In Proceedings of SIGIR-97, 20th ACM International Conference on Research and Development in Information Retrieval. Philadelphia, PA, 1997: 67-73.

[5] Lewis D D. An evaluation of phrasal and clustered representations on a text categorization task. in Proceedings of the 15th International ACM SIGIR Conference on Research and Development in Information Retrieval. Copenhagen, Dk: ACM Press. 1992: 37-50.

[6] Lewis D D, Ringutte M, Comparison of two learning algorithms for text categorization, In: Proceedings of the Third Annual Symposium on Document Analysis and Information Retrieval (SDAIR'94), 1994.

[7] Tom Mitchell. Machine Learning, McCraw Hill, 1996.

[8] Lewis D, Shapire R E, Callan J P, Papka R. Training Algorithms for Linear Text Classifiers. Proc. 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. 1996: 298-306.

[9] Moulinier I, Raskinis G, Ganascia J. Text categorization: A symbolic approach, In: Proceedings of the Fifth Annual Symposium on Document Analysis and Information Retrieval, 1996.

[10] Quinlan J R. Induction of decision trees, Machine Learning, 1986, 1(1): 81-106.

[11] Wiener E, Pedersen J Q, Weigend A S. A neural network approach to topic spotting, In: Proc. 16th Ann. Int. ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'93), 1993: 22-34.

[12] Ng H T, Goh W B, Low K L. Feature selection, perceptron learning, and a usability case study for text categorization. In: 20th Ann. Int. ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'97), 1997: 67-73.

[13] Ruiz M E, Asan P S, Hierarchical Text Categorization Using Neural Networks, Information Retrieval, 2002, 5, 87-118.

[14] Yang Y, Chute C G. An example-based mapping method for text classification and retrieval. (pdf. gz) ACM Transactions on Information Systems (TOIS) 1994, 12(3): 252-77.

[15] Vapnik V N. The Nature of Statistical Learning Theory. New York: Springer-Verlag, 1995.

[16] Joachims T. Text Categorization with Support Vector Machines: Learning with Many Relevant Features. In ECML-98, 1998.

[17] Tong S, Koller D. Support Vector Machine Active Learning with Applications to Text Classification. Journal of Machine Learning Research, 2001, 2: 45-66.

[18] Omprakash D Gnawali. A Keyword-set Search System for Peer-to-Peer Networks. MIT's thesis Lib. 2002, 6.

[19] Gnutella. http://gnutella. wego. com.

[20] Stokes M. Gnutella2 Specifications Part One. http://www. gnutella2. com/gnutella2 search. htm.

[21] Kalogeraki V, Gunopulos D, Zeinalipour-Yazti D. A Local Search Mechanism for Peer-to-Peer Networks. In CIKM, 2002.

[22] Tsoumakos D, Roussopoulos N. Adaptive Probabilistic Search (APS) for Peer-to-Peer Networks. Technical Report CS-TR-4451, Un. of Maryland, 2003.

[23] Crespo A,Garcia-Molina H. Routing Indices for Peer-to-Peer Systems. In ICDCS,2002,7.
[24] Yu B,Singh M P. Searching social networks. In Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS). ACM Press,2003,7.
[25] Stoica I,Morris R,Karger D,Kaashoek M F,Balakrishnan H. Chord: A scalable peer-to-peer lookup service for internet applications. ACM SIGCOMM,2001.
[26] Zhao B,Kubiatowicz J,Joseph A. Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. Technical Report,UCB/CSD-01-1141,2000,4.
[27] Rowstron A,Druschel P. Pastry: Scalable,distributed object location and routing for large-scale peer-to-peer systems. Lecture Notes in Computer Science,2001,2218: 329-350.
[28] Reynolds P. Vahdat A. Efficient Peer-to-Peer Keyword Searching. In Unpublished Manuscript, 2002,6.
[29] Bloom B H. Space/Time Tradeoffs in Hash Coding with Allowable Errors. Communications of the ACM,1970,713(7): 422-426.
[30] Bobby Bhattacharjee, Sudarshan Chawathe, Vijay Gopalakrishnan, Pete Keleher, Bujor Silaghi. Efficient peer-to-peer searches using resultcaching. in The 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03),2003,2.
[31] Bryce Wilcox-O'Hearn. Experiences deploying a large-scale emergent network. In Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS'02),Cambridge,MA,2002,3.

CHAPTER 5

第5章 网格计算研究与进展

网格计算(Grid Computing)的产生和发展为人们提供了透明访问和使用分布、异构、动态资源的方式,同时也给分布式计算的研究领域带来了许多机遇和挑战。如何高效、合理地对网格资源进行管理与调度、使得大规模的科学计算问题能够得到可靠的解决,已经成为一个重要的研究热点,这也是本书的研究目的。

5.1 网格简介

网格的概念是借鉴电力网(Electric Power Grid)概念提出的,网格计算的最终目标是实现用户在使用计算力时,能够如同当前使用电能一样方便。当人们在使用电能时,无须知道它的来源地以及产生的方式,只要知道它是一种统一形式的电能就可以了。类似的,通过网格计算,用户希望在使用计算力时,无须关心计算力的来源地以及计算设施的形式,计算力以统一的形式呈现在用户面前。网格计算代表了一种先进的技术和基础设施,是分布式高性能计算和高吞吐量计算的主要潮流和今后的发展方向。

5.1.1 网格的分类

根据网格所共享的资源和所支持的应用,可以做如下分类:

1. 计算网格

这类网格利用分散在广域网上的动态共享计算资源进行科学工程计算以解决各种研究问题,这是网格最初的应用模式。进一步,计算网格包括分布式高性能计算、高吞吐量计算。分布式高性能计算利用广域网上的计算资源使单个应用程序的执行时间最短。而高吞吐量计算使整个网格系统的利用率达到最大。

2. 抽取网格

这类网格从空闲的服务器和台式机上抽取 CPU 时间片,以解决资源密

集型的任务计算。

3. 数据网格

这类网格动态共享分散在广域网上的数据资源以解决海量存储问题，如数字图书馆、数据仓库等。

4. 服务网格

这类网格提供了任何单个机器都不能提供的服务，例如点播服务、协作服务、多媒体网格等。协作服务提供了虚拟工作空间，使用户和程序实时交互。

5.1.2 国内外网格研究项目

网格计算从研究中心和学校开始，现在一些商业企业也在使用网格。网格计算开创了一种新的金融和商业模式。在金融服务领域，网格计算可提高贸易交易的速度，处理大量的数据，提供更加稳定的 IT 环境，减少启机时间。政府代理机构可以使用网格存储、保护和集成巨大的库存数据。许多民用和军用的代理机构对跨部门的协作、数据的同一性和安全性等方面都有巨大的需求，这也可以通过网格来实现。在生命科学领域，公司可以使用并行的网格计算处理大量的数据，加快数据的处理就意味着可以快速占有市场。在这一行业，极细微的因素都是决定性的。网格计算的重要性，不言而喻，统计数据表明，主机系统有 40％的时间空闲，UNIX 服务器实际的工作时间只有 10％，正常情况下，一天中 95％的时间 PC 机什么都没做。借助于网格，可以把这部分资源糅合在一起统一使用。

在过去的几年时间内，网格计算迅速成为全球计算机界关注的热门技术之一，得到了许多国家学术界、企业界甚至政府部门的高度重视和支持。美国、英国、日本、加拿大、意大利、澳大利亚等国家政府相继投入巨资启动了庞大的网格研究与开发计划。例如美国的 TeraGrid、NEESGrid 和著名的 Globus 项目，英国国家网格 U. K. National Grid，德国的计算资源统一接口项目 Unicore，以及亚太地区网格 APGrid 等。

我国在网格计算方面也做了大量基础性和前瞻性的研究，例如，科技部以“高性能计算机及其核心软件”863 专项课题的形式，在“十五”期间大力支持网格的研究和应用开发工作。国家自然科学基金委迅速成立“e-science 科学研究活动环境”重大专项。教育部依托教育与科研网 CERNET 和高校的大量计算资源和信息资源，推出 ChinaGrid 计划。上海市也于 2003 年 11 月份启动了上海市科委重大科研攻关项目“信息网格及其典型应用研究”，建立具有上海特色的信息网格，率先推进城市网格计划。

5.2 网格资源管理和调度策略研究现状

5.2.1 资源管理系统

总体来讲，现有的资源管理系统可以分成三种：集中式资源管理模式、分布式资源管理模式和层次式资源管理模式(或称混合式管理模式)。

1. 集中式资源管理模式

集中式资源管理系统可以用来管理单一或者多种资源，尤其适合于集群系统的管理，例

如 Condor[1]、LSF[2]、Condine[3]等系统均属于集中式的资源管理系统。集中式的管理模式具有以下优点：结构简单、易于管理、可以确保一致性和完整性。但是，由于调度瓶颈问题，集中式资源管理很难用于分布式系统的资源管理。

2. 分布式资源管理模式

分布式资源管理系统将资源分成多个不同的虚拟组织(Virtual Organization, VO)[4]进行管理，每个虚拟组织内部都存在一个域调度器。这种管理模式具有很好的扩展性，但是难以获得其他域的状态，因此，整个系统的最佳调度性能存在问题。此外，不同域之间的通信量很大，由于数据是分散的，不能保证在多个域之间进行调度时数据的一致性。

3. 层次式资源管理模式

层次式资源管理模式集合了集中式和分布式资源调度模式的优点，不仅能够消除两种模型的缺点，同时还能够解决分布式系统资源管理的一些挑战性问题，例如，系统的自治性、异构环境和调度策略的扩展性等。这种模式非常适合用于网格系统的资源管理，Globus、Legion[5]、Ninf[6]、NetSolve[7]等系统都采用这种层次式的混合资源管理模式。

5.2.2　网格资源管理系统

在网格环境中，存在各种不同类型的异构资源，包括计算资源、存储资源、网络资源等，在这里资源是指具有特定功能的网络化实体，同时存在种类繁多的需求资源的网格应用。作为提供资源的资源提供者和需求资源的资源消费者，它们有着各自使用资源的策略、目的。根据网格用途，从资源使用的角度可以将网格分成为专用网格、通用网格。

在传统的计算网格中，系统中的资源专属于某一组织，例如，美国 NASA 的信息力网格(Information Power Grid)[8]仅供 NASA 机构内的科研人员使用，通常采用集中式方法控制，从而能够保证网格服务的质量。

通用网格是专用网格的扩充，是应大量存在的短期协作的需求而构建的。例如，在地理上分布的一组科学家需要开展短期的协作和资源共享以完成热化学应用仿真实验，这就需要组成一个虚拟组织，定义 VO 中每个参与者的权利，建立相互之间的认证。同时每个参与方需要支持网格中间件，以及定义它们在中间件之上向虚拟组织的其他成员提供的服务。在这种情况下，分散管理和分层管理是一种有效的管理方式。通用网格资源管理的模式还可以从相互之间协作模式的资源共享做进一步扩展，即通过一定的激励，使得人们愿意贡献出其空闲的计算资源，进而构造一个公共网格和普适网格[9]，如同电力网一样，需求资源的用户，能如同从电力网中使用电力一样很方便地从网格中得到计算力。

无论是专用网格系统，还是通用网格系统，其资源管理系统的基本功能是接受网格上用户的请求，根据当前可用资源信息以及用户请求的服务质量(QoS)，将用户的请求映射到最佳资源组合上。一个资源管理系统至少提供三种基本服务：资源信息发布、资源发现机制和资源调度[10]。

目前众多的网格计算研究项目，分别有其各自的资源管理系统，抽象其具体实现，可以得到以下属性[10]：机器组织、资源描述模型、命名空间组织、服务质量、资源信息存储方式、资源发现机制、调度组织、调度策略、状态估计、再调度(Rescheduling)等。

根据以上资源管理属性，下面分别对两种网格核心层中间件 Globus 和 Legion，以及三

种典型的实现资源管理与调度的用户层中间件 AppLes、Conder-G 和 Nimrod-G 的资源管理系统进行分析。

1. Globus 资源管理中间件

在 GT 2.x 中，Globus 的机器组织采用层次结构，上层的服务可以由多个下层服务来实现[11]，通过基于 LDAP 元计算目录服务(MDS)[12]提供当前资源信息。其中 MDS 提供网格索引信息服务(GIIS)和网格资源信息服务(GRIS)，GRIS 实现了一个统一的接口，通过这个接口，可以查询网格中当前资源提供者的信息，包括它们当前的配置、计算能力及状态。GIIS 通过从众多 GRIS 中“pull”信息构造一个单一一致的资源信息数据库，而资源提供者通过“push”协议更新它所属的 GRIS。MDS 命名采用树结构，Globus 以资源预留方式提供 QoS，Globus 提供调度组件但不提供调度策略。高层的资源代理(Resource Broker)通过 LDAP 协议查询 MDS 获得资源信息，目前这样的资源代理有 AppLes、Condor-G 和 Nimrod-G 等。GT 4.x 基于 Web Service，但仍采取层次结构组织资源，采用 GIIS 管理资源信息，而 GRIS 则包含在 OGSI(Open Grid Services Infrastructure)核心框架中。OGSI 制订了一组适用于所有网格服务的服务元语。更多的规范也正在制订中。

2. Legion 资源管理中间件

Legion[5]借用了面向对象的概念和管理，使得系统中的任何一个实体都被看作一个自主管理并有自己的控制流的对象，实体之间的通信则赋予对象方法调用的概念。它采取层次化系统结构，命名空间采用图模式，提供软服务质量(soft QoS)，目录服务采用联合文件系统(Federated File System)，采用定期 pull 实现资源信息发布，基于查询实现资源发现。Legion 提供两种类型的资源，分别是计算资源和存储资源。Legion 的调度基于两个原则：

(1) 地点自治，即资源提供者对如何使用其拥有的资源有最终的决定权；

(2) 用户自治，即用户自己可以选择调度策略从而为改善性能提供了可能。

3. AppLes

AppLes(Application Lever scheduler)[13,14]为每个独立的网格应用程序提供一个调度代理(Agent)，AppLes Agent 利用 NWS(Network Weather Service)提供的静态和动态的应用程序以及系统的信息来为其对应的应用程序分配一组可用的资源，与底层的 Globus 或 Legion 的资源管理系统交互完成任务的执行。AppLes 采用分散式调度、预测启发式状态估计、在线再调度、以应用程序为中心的调度策略，不提供 QoS。为便于软件重用[15]，AppLes 进行了改进，调度由两个进程 client 和 daemon 完成，daemon 完成配置，client 则与用户交互。

4. Condor-G

Conder[16,1]是一个高吞吐量计算环境，用来管理集群系统，已被扩展到网格环境 Conder-G[17]。在 Condor 中，采用集中式调度，用户代理管理由应用程序描述所组成的对列，负责向匹配器发出请求，资源代理实现资源所有者策略，并向收集器(Collector)提供资源信息。匹配器接受用户代理请求，并向收集器查询资源信息，完成资源请求与资源提供者之间的匹配，然后由匹配的用户代理在相应的资源上启动计算。它采用平面结构的机器组织、ClassAd 语言的资源描述、分层关系的命名空间、不提供 QoS、集中式查询资源发现机制

以及定期 push 资源信息。

5. Nimord-G

Nimord-G[18,19]是一个网格资源代理，基于底层 Globus 或 Legion 提供的服务，在计算网格上管理和控制参数扫描程序。资源代理负责处理用户请求，收集当前资源信息，转化请求为任务并调度任务到相应的资源上，并收集计算结果返回给用户。Nimord-G 采用层次结构、资源经济模型、分散式调度、预测价格模型、事件驱动再调度、以应用程序为中心的调度策略。

5.2.3 网格环境下的资源调度策略

以上主要论及构建一个网格资源管理系统，而如何在已有的网格基础设施之上利用底层提供的功能进行有效的资源管理和调度是另一个重要的研究课题。

并行计算中，调度问题包括两类——作业调度（Job Scheduling）和任务调度（Task Scheduling）。作业调度，又称高吞吐量调度（High Throughput Scheduling），是指调度多个相互独立的任务，使这个系统的利用率最高。而任务调度，又称高性能调度（High Performance Scheduling），是指调度单个并行程序中多个具有数据依赖关系的任务，使程序的执行时间最短。节文研究作业调度。

调度有静态调度（Static Scheduling）和动态调度（Dynamic Scheduling）两种情形。当并行程序的属性，如任务的执行时间、通信和同步、任务依赖关系等，能够事先获得，而且目标系统是专用的，那么可以在编译时完成静态调度，此时称为编译时静态调度（Compile-Time Static Scheduling）；如果目标系统是多用户共享的，系统特性随时间变化，那么调度必须在运行时完成，当调度生成后不再变化，程序按照调度方案完成，此时称为运行时静态调度（Run-Time Static Scheduling）；如果是根据系统的变化，不断修改调度方案，则称为运行时动态调度（Run-Time Dynamic Scheduling），简称动态调度。需要说明的是，运行时静态调度与运行时动态调度之间的区别并不是很严格，通常将运行时静态调度方案作为动态调度方案的基础。一般来讲，由于各自优化的目标不同，作业调度通常采用动态调度。

一个网格应用程序包括多个相互通信和协作的任务，调度的目的是将这些任务适当地安排到资源上以最有效的方式执行。调度应用程序设计了一系列活动：

（1）选择资源；

（2）将任务分派到计算资源上；

（3）分布数据或者重定位数据和计算；

（4）在计算资源上排序任务；

（5）对任务间通信排序。

上述步骤（1）通常称为资源选择（Resource Selection）、资源发现（Resource Discovery）。资源选择是指从资源池中选取候选资源的过程；而资源发现则是确定系统中那些资源可以供应用程序使用。步骤（1）～（3）（称为 Mapping）主要在空间上分配计算和数据，步骤（4）～（5）（称为 Scheduling）则在时间上分配计算和通信。两者统一称为调度。不失一般性，本书将 Mapping 称为网格资源环境下的广域调度，而 Scheduling 则称为本地资源调度，本书研究广域调度，即 Mapping 过程，对应步骤（1）～（3）。

针对传统计算网格,即专用网格,网格环境下的调度算法目前主要集中在扩展早期在异构环境下独立任务调度的研究,将元任务(Meta-Task)[20]或任务包[21]调度到网格中不同资源域(Resource Domain)或集群(Cluster),这是一个 NP 问题[22],因此目前主要利用启发式算法来解决此类问题。

早期异构环境启发式算法有:随机调度任务到下一个可用机器上的 OLB(Opportunistic Load Balancing)[23~25],调度任务到最佳执行时间机器上的 MET(Minimum Execution Time)[23,24],调度任务到最小执行时间机器上的 MCT(Minimum Completion Time)[23]、Min-min[23,24,26]、Max-min[23,24,26]、SA(Switching Algorithm)[27,28]、Sufferage[26]、GA(Genetic Algorithms)[29~31]、GSA(Genetic Simulated Annealing)[32,33]、Duplex[23,24,34,35]等。

参考文献[36]给出了利用 SA 算法在网格环境中进行任务调度的方法;参考文献[37]给出在计算网格中调度任务组的算法;考虑到网格中机器的动态性,复制多份任务到网格的调度算法在参考文献[38]中给出;参考文献[14,39]将 Min-min、Max-min、Sufferage 算法扩展到网格环境,并提出了 Xsufferage 算法;参考文献[40]结合 QoS 到 Sufferage 中,提出了改进的 Sufferage 算法;参考文献[41]给出了计算网格中基于期限的客户/服务系统的调度算法。参考文献[42]给出 GA 算法在网格调度上的应用。参考文献[43]对网格中各种调度策略做出了评价。以下对其中几种比较典型的网格任务调度算法做简单的分析。

1. MCT 和 MET 算法

MCT 算法的主要思想是分配任务给完成时间最早的处理器,这种算法可能会把一些任务分配给执行时间不是最短的处理器。但是,由于算法根据最早完成时间来分配任务,从而可以尽量使得处理器间的负载均衡分布。

MET 算法的主要思想是分配任务给执行时间最短的处理器,该算法的最大优点是非常简单。与 MCT 算法相比,MET 算法没有考虑处理器的就绪时间,因此可能会把很多任务分配给部分处理器,从而导致处理器之间的负载不均衡。

2. SA 算法

SA(Switching Algorithm)算法根据处理器的负载分布情况循环使用 MCT 和 MET 算法。假定所有处理器的最大准备时间是 r_{max},最小准备时间是 r_{min},那么处理器之间的负载均衡因子用 $\pi = r_{min}/r_{max}$ 来表示,参数 π 取[0,1]之间的任意值。两个阈值 π_{low} 和 π_{high} 在[0,1]区间被选择,并且满足条件 $\pi_{low} < \pi_{high}$。初始化时,设置 π 值为 0。SA 算法开始用 MCT 来映射任务,在负载均衡因子增加到 π_{high} 以后,改用 MET 算法来映射任务。这样负载均衡因子会开始下降,当下降到 π_{low} 以后,SA 算法开始使用 MCT 算来映射任务,如此循环下去。SA 调度算法有随网格资源提供者的资源属性动态变化而动态改变调度策略的自适应性,但因为同时需要测定两个阈值,并且需要根据均衡因子切换调度算法,这无疑增加了额外开销。

3. Min-min 算法

算法首先计算当前元任务中每一个子任务的最小完成时间,然后将具有最小完成时间的子任务分配给相应的处理器,同时更新相应的处理器的就绪时间,被分配的子任务从元任务移去,如此重复分配元任务,直至整个任务分配完成。该算法使得处理器的就绪时间状态信息变化最小。如果子任务 t_i 和 t_k 竞争处理器 h_j,Min-min 算法会把对处理器 h_j 的就绪

时间做相对较小改变的任务 t_k 分配给 h_j。这样，增加了 t_k 在处理器 h_j 上有最早完成时间的可能性。因为 $t=0$ 时，处理器完成时间最早的任务就是它执行最快的任务，如果分配给处理器的大批量任务不仅完成时间短而且执行快，则获得的吞吐量将相对较小。

4. Max-min 算法

与 Min-min 算法类似，一旦确定了处理器对每个任务的最早完成时间，具有最大的最早完成时间的任务 t_k 将被分配给相应的处理器。时间矩阵 C 和处理器就绪时间向量 r 被更新，对于剩下的任务进行重复处理。在短任务多，长任务少的情况下，Max-min 算法的性能将会不错。例如，如果网格系统中仅含有一个长任务，那么这个长任务可以与多个短任务并发执行，这样元任务的最大完成时间由最长任务的完成时间决定。

5. DGS 算法

DGS(Dynamic Grouping Scheduling)算法的最大特点在于它采用了一种新的动态任务分组策略。这种任务分组策略对于网格计算环境中的主机，根据其计算能力依次编号，对于计算任务按照负载大小依次编号，并把计算任务根据其负载大小与主机的计算能力相对应地分成 $|H|$ 组。如果一个任务的编号比某个主机的编号小，那么这个主机就能执行这个任务。这样保证了一个计算能力低下的主机不会执行一个计算量很大的任务。同时，任务的所属组别可以随资源属性的变化而动态改变。DGS 算法主要利用 DAG(有向无环图)进行任务映射，并且在异构环境中没有考虑通信开销，算法的时间复杂度为 $O(|T|^3)$。

以上算法通常都基于这样的假设，即每个任务的执行时间，所需的数据在调度之前是可知的，然后利用启发式算法将任务调度到网格中的不同资源域中。还有另外一种方法，即认为网格中的资源信息很少，应用程序的执行情况事先难以预测，因此采用将一个任务复制多份，由调度器随机映射到不同的资源域，第一个返回的结果将作为该应用程序的结果，同时终止尚在进行的程序拷贝。

相对于以上针对传统计算网格的调度策略和算法而言，还有另外一类策略与算法，即在资源管理中基于经济学中的相关理论(如一般均衡理论、拍卖理论等)，在通用网格中实现有效的资源管理。网格资源管理的经济模型可以分为[44]商品市场模型、标价模型、议价模型、合同网模型、拍卖模型等，其中以商品市场模型与拍卖模型应用最多，以下对这两个经济模型做简单介绍。

6. 拍卖模型

拍卖模型通过多个资源消费者以不同的价格竞争同一资源，最后由拍卖者依据不同的准则选定投标人并确定资源成交的价格。这种拍卖模型有四种基本类型：上升出价拍卖(Ascending-Bid Auction)，又称英国式拍卖(English Auction)；下降出价拍卖(Descending-Bid Auction)，又称荷兰式拍卖；第一价格暗标拍卖(First-Price Sealed-Bid Auction)；第二价格暗标拍卖(Second-Price Sealed-Bid Auction)，又称 Vickrey 拍卖。另外还衍生出很多复杂的拍卖机制，例如多个商品拍卖问题(Multi-Unit Auctions)、多个买方和多个卖方的双向拍卖(Double Auctions)等。

7. 商品市场机制

不同于拍卖机制，商品市场模型主要侧重于同种商品。在商品市场中，不同商品在市场上进行交换和消费，不同商品在市场上的活动是相互联系和影响的，因此存在一个问题，即

所有商品的供给与需求能否同时达到平衡，这即是经济学上的一般平衡理论。目前将商品市场模型运用到网格资源管理的典型代表有 G-commerce 和 GRACE。在 G-commerce[45] 中仅考虑 CPU 和存储空间，均衡价格采用近似均衡，为避免收集全局供求信息，超额需求函数由多项式拟合。在 GRACE[46] 中，在基于经济学原理的资源调度实验中，资源价格是根据资源的重要性事先指定，因此难以保证其资源配置分配决策的最优性[47]。

商品市场模型在具体应用到网格资源管理中时存在以下难点：模型中效用函数的确定；均衡价格的确定。均衡价格在经济学中通过 Tatonnement 过程调整价格而得到，因此需要指定高效的计算均衡价格算法。在确定均衡价格时，需要收集全局超额需求信息，因此理论上需要全局信息。均衡价格下的配置是不是最优配置，需要进一步限定其成立的条件。

5.2.4 现有研究的不足与分析

目前的网格资源管理与调度可以归为两种模式：集中控制与分散控制。集中控制是由一个资源管理器来统一管理资源的分配和调度，如 Globus、Legion 等，为适应在大范围的有效资源管理，通常集中控制采取分层方式。分散控制(如 AppLes)为每个应用提供一个负责调度的代理。前者的扩展性存在局限，并且由于存在调度瓶颈，使得其很难用于分布式系统的资源管理。后者具有很好的扩展性，但是因为网格资源的动态性以及各个应用调度器各自调整负载导致资源分配的复杂化，因此稳定性不好；同时由于难以获得全局的状态，因此，整个系统的最佳调度性能存在问题；此外，不同域之间的通信量很大，由于数据是分散的，不能保证在多个域之间进行调度时数据的一致性。调度算法研究也主要集中在两个方向：基于启发式算法的调度和基于经济模型的调度。不同的启发式调度算法分别具有各自的优缺点，例如 MET 算法不考虑处理器的就绪时间，因此可能会把很多任务分配给部分处理器，从而导致处理器之间的负载不均衡；SA 调度算法虽然有随网格资源提供者的资源属性动态变化而动态改变调度策略的自适应性，但因为同时需要测定两个阈值，并且根据均衡因子切换调度算法，无疑增加了额外开销；Min-min 认为处理器完成时间最早的任务就是它执行最快的任务，因此如果分配给处理器的大批量任务不仅完成时间短而且执行快，则可能仅仅获得相对较小的吞吐量；Max-min 在短任务多、长任务少的情况下执行性能不错；等等。另外一方面，经济模型在具体应用到网格资源管理和调度中时也存在难点，例如，效用函数的确定、均衡价格的确定。在网格环境中需要指定高效的计算均衡价格算法，理论上需要收集全局超额需求信息；另外，均衡价格下的配置是不是最优配置，同样需要进一步限定其成立的条件。

5.3 本章小结

本书网格计算部分在分析了网格资源的研究内容和目前存在的研究问题的基础上，重点研究了以下一些问题：

(1) 何种体系结构能够更好地反映网格资源提供者的动态性、自治性和异构性，能够提高网格系统的鲁棒性和自适应性，并且能够支撑在此基础上的资源管理和调度等问题的后续研究。

(2) 如何进行高效的网格资源管理与调度，实现网格系统的动态调度和负载均衡。研究重点包括设计适合于不同种类网格系统，如专用计算网格和基于 Web 服务的通用网格的资源调度策略，以解决现有调度策略存在的一些问题，提高调度性能。

为此，本书研究了相关的模型和算法，主要包括对等网络模型、多代理模型及其相关算法，以及服务发布与服务发现的算法等，并通过大量的理论分析和实验验证提出的模型和算法的有效性。

由于网格系统的一个重要特性就是它的动态性和自治性，即计算资源提供者隶属于不同的域、呈现 P2P 的特点。在网格环境中，异构的计算资源，例如个人计算机、大型集群、在线设备等可能属于不同的组织，必须通过各种共享策略使之成为网格系统中的可利用资源。网格计算的动态性表现在很多方面：静态因素如操作系统版本、存储空间等；高度动态变化的因素包括可利用 CPU、工作负载以及网络带宽等。网格资源管理方法必须考虑这些动态可变的因素。如何充分考虑这些动态的因素，利用有效的管理机制进行资源调度，实现负载均衡，是网格资源管理要研究的重要内容。因此，贯穿本文的一个主要技术路线就是如何利用有效的管理机制，充分考虑网格中动态的因素，从而能实现网格资源的实时动态调度和负载均衡，这体现在下面列出主要研究内容和创新性工作中。

(1) 首先，在分析了网格自身特点和发展趋势的基础上，将对等网络方法引入网格的资源管理和调度，设计了基于超级结点对等网络的网格资源管理体系结构。这种集中式和分布式的混合结构设计，能够解决现有网格系统采用集中式管理容易引起的单点失效、性能瓶颈等问题，从而可以更好地描述网格资源的动态性、自治性等特点，使网格系统具有更强的鲁棒性和自适应性，并且有利于制定优化网格资源管理和调度的策略、算法。进一步，根据网格资源提供者的 IP 层信息生成含有路由信息的 Overlay Network 拓扑，并且使用有向图表示该拓扑结构。这种使用有向图进行网格拓扑结构表示的方式在能够准确描述网格资源提供者的计算能力的同时，还能够弥补其他现有的资源信息表示模型的 Overlay 层路由信息不能精确反映 IP 层路由情况的不足，同时这种简单的描述方式利于网格资源调度器发掘网格资源提供者和网格任务之间的对应关系。

(2) 针对为特定的科学应用而设计的专用计算网格的资源调度，提出了基于树匹配的 nTreeMatch 算法。算法结合 DAG(有向无环图)的任务表示形式，通过树形数据结构匹配的方法解决了网格资源和网格任务间的映射问题。同时算法充分利用 Overlay 拓扑中结点的路由信息，以轻量附加开销来有效减少 Overlay 层上的路由跳数，使得 Overlay 层上的路由跳数尽量接近 IP 层上的路由跳数，降低 RDP(相对平均延迟开销)。理论和模拟实验表明在大规模的网格系统中，算法在进行资源调度时可以获得较高的路由效率，为路由的状态与效率折中问题提供了一个可行的解决方案。

(3) 针对基于 Web Service 的通用网格系统的资源调度，提出了基于资源发现的 GChord 算法。考虑到网格的动态性特征，GChord 算法采用服务发现的方式解决资源调度问题，将资源需求按照 Chord 路由协议在网格中转发，改变了传统的集中式调度方法采取的信息收集方式，能够实时反映网格结点的工作负载状态，有效解决由于信息过时、数据不一致而引起的任务再调度问题。实验证明，GChord 算法可以实现网格系统的实时资源调度，并且使得网格系统保持良好的负载均衡状态。

(4) 在研究了多代理技术和网格计算相互融合的发展趋势的基础上，为解决网格资源

调度中分散调度及动态负载均衡的挑战，本书提出了基于多代理协同计算的 rwAgent 算法。算法利用多代理技术，通过代理的自治性和智能学习，实现网格资源的分散调度，同时可以获得很好的负载均衡效果。严格的数学建模和理论分析证明，rwAgent 算法可以实现资源调度过程中网格系统的全局负载均衡。实验结果证明了算法的有效性和优越性。

参考文献

[1] Basney J，Livny M. Deploying a High Throughput Computing Cluster. In：R. Buyya（editor），High Performance Cluster Computing：Architectures and Systems，Vol. 1，Prentice Hall PTR，1999. 116-134.

[2] Zhao Q，Suzuki J. Efficient quantization of LSF by utilizing dynamic interpolation. In：IEEE International Symposium on Circuits and Systems，Hong Kong，1997.

[3] Ferstl F. Job and resource management systems. In：Architectures and Systems，Vol. 1，1999. 499-518.

[4] Foster I，Kesselman C，Tuecke S. The Anatomy of the Grid：Enabling Scalable Virtual Organizations. In：International Journal of Supercomputer Applications，Vol. 15(3)，2001. 200-222.

[5] Chapin S，Katramatos D，Karpovich J，Grimshaw A. Resource Management in Legion. In：Future Generation Computer Systems，Vol. 15(5)，1999. 583-594.

[6] Nakada H，Sato M，Sekiguchi S. Design and Implementations of Ninf：towards a Global Computing Infrastructure. In：Future Generation Computing Systems：Metacomputing Special Issue，1999.

[7] NetSolve Project. http://icl. cs. utk. edu/netsolve.

[8] Johnston W，Gannon D，Nitzerg B. Grids as Production Computing Environments：The Engineering Aspects of NASA's Information Power Grid. In：the 8th IEEE International Symposium on High Performance Distributed Computing，1999. 197-204.

[9] Von Laszewski G，Blau E，Bletzinger M，Gawor J，Lane P，Martin S，Russell M. Software，Component，and Service Deployment in Computational Grids. In：the IFIP/ACM Working Conference on Component Deployment，LNCS，Vol. 2370，2002. 244-256.

[10] Krauter K，Buyya R，Maheswaram M. A Taxonomy and Survey of Grid Resource Management System. In：Software - Practice and Experience，Vol. 32(2)，2002. 135-164.

[11] Czajkowski K，Foster I，Karonis N，Kesselman C，Martin S，Smith W，Tuecke S. A Resource Management Architecture for Metacomputing Systems. In：the IPPS/SPDP Workshop on Job Scheduling Strategies for Parallel Processing，LNCS，Vol. 1459，1998. 62-82.

[12] Czajkowski K，Fitzgerald S，Foster I，Kesselman C. Grid Information Services for Distributed Resource Sharing. In：the 10th IEEE International Symposium on High-performance Distributed Computing，IEEE Computer Society Press，2001. 181-194.

[13] Berman F，Wokski R. The AppLeS Project：A Status Report. In：the 8th NEC Research Symposium，1997.

[14] Casanova H，Obertelli G，Berman F，Wolsi R. The AppLeS Parameter Sweep Template：User-level Middleware for the Grid. In：Scientific Programming，Vol. 8(3)，2000. 111-126.

[15] Casanova H，Berman F. Parameter Sweeps on the Grid with APST. In：F. Berman，G. Fox，T. Hey (editor)，Grid Computing：Making the Global Infrastructure a Reality，Wiley Publisher，Inc，2002. 773-787.

[16] Litzkow M，Livny M，Mutka M. Condor：a Hunter for Idle Workstations. In：the 8th International Conference on Distributed Computing Systems，IEEE Computer Society，1998. 104-111.

[17] Frey J，Tannenbaum T，Foster I，Livny M，Tueche S. Condor-G：A Computation Management Agent

for Multi-Institutional Grids. In: Cluster Computing, Vol. 5(3), 2002. 237-246.
[18] Buyya R, Abramson D, Giddy J. Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid. In: the 4th International Conference on High-Performance Computing in Asia-Pacific Region, IEEE Computer Society, 2000. 283-289.
[19] Abramson D, Buyya R, Giddy J. A Computational Economy for Grid Computing and Its Implementation in the Nimrod-G Resource Broker. In: Future Generation Computer System, Vol. 18 (8), 2002. 1061-1074.
[20] Maheswaran M, Ali S, Siegel H, Hensgen D, Freund R. Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems. In: Parallel and Distributed Computing. Vol. 59(2), 1999. 107-131.
[21] Kebbal D, Talbi E, Geib J. Building of Scheduling Parallel Adaptive Applications in Heterogeneous Enviornments. In: the 1st IEEE International Workshop on Cluster Computing, IEEE Computer Society, 1999. 195-201.
[22] EI-Rewini H, Lewis T, Ali H. Task Scheduling in Parallel and Distributed Systems. Prentice Hall, 1994.
[23] Armstrong R, Hensgen D, Kidd T. The Relative Performance of Various Mapping Algorithms is Independent of Sizable Variances in Run-time Predictions. In: the 7th IEEE Heterogeneous Computing Workshop, IEEE Computer Society, 1998. 79-87.
[24] Freund R, Gherrity M, Ambrosius S, Camplell M, Halderman M, Hensgen D, Keith E, Kidd T, Kussow M, Lima J, Mirabile F, Moore L, Rust B, Siegel H. Scheduling Resource in Multi-user, Heterogeneous Computing Environment with SmartNet. In: the 7th IEEE Heterogeneous Computing Workshop, IEEE Computer Society, 1998. 184-199.
[25] Freund R, Siegel H. Heterogeneous Processing. In: IEEE Computer, Vol. 26(6), 1993. 13-17.
[26] Maheswaran M, Ali S, Siegel H, Hensgen D, Freund R. Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems. In: Journal of Parallel and Distributed Computing, Vol. 59(2), 1999. 107-131.
[27] Coli M, Palazzari P. Real Time Pipelined system Design Through Simulated Annealing. In: Journal of Systems Architecture, Vol. 42(6-7), 1996. 465-475.
[28] Zomaya A, Kazman R. Simulated Annealing Techniques. In: M. Atallah (editor), Algorithm and Theory of Computation Handbook, CRC Press, 1999. (37-1)-(37-19).
[29] Singh H, Youssef A. Mapping and Scheduling Heterogeneous Task Graphs Using Genetic Algorithm. In: the 5th IEEE Heterogeneous Computing Workshop, IEEE Computer Society, 1996. 86-97.
[30] Srinivas M, Patnaik L. Genetic Algorithm: A Survey. In: IEEE Computer, Vol. 27(6), 1994. 17-26.
[31] Wang L, Siegel H, Roychowdhury V, Maciejewski A. Task Matching and Scheduling in Heterogeneous Computing Environment Using a Genetic-Algorithm-Based Approach. In: Journal of Parallel and Distributed Computing, Vol. 47(1), 1997. 8-22.
[32] Chen H, Flann N, Watson D. Parallel Genetic Simulated Annealing: A Massively Parallel SIMD Algorithm. In: IEEE Transaction on Parallel and Distributed Systems, Vol. 9(2), 1998. 126-136.
[33] Shroff P, Watson D, Flann N, Freund R. Genetic Simulated Annealing for Scheduling Data-dependent Tasks in Heterogeneous Environments. In: the 5th IEEE Heterogeneous Computing Workshop, IEEE Computer Society, 1996. 98-104.
[34] Kafil M, Ahmad I. Optimal Task Assignment in Heterogeneous Distributed computing Systems. In: IEEE Concurrency, Vol. 6(3), 1998. 42-51.
[35] Shen C, Tsai W. A Graph Matching Approach to Optimal Task Assignment in Distributed Computing

System Using a Mini-Max Criterion. In: IEEE Transactions on Computers, Vol. 34 (3), 1985. 197-203.

[36] Yarkihan A, Dongarra J. Experiments with Scheduling Using Simulated Annealing in a Grid Environment. In: the 3rd International Workshop on Grid Computing, LNCS, Vol. 2536, 2002. 232-242.

[37] Chen H, Maheswaran M. Distributed Dynamic Scheduling of Composite Tasks on Grid Computing System. In: the International Parallel and Distributed Processing Symposium: IPDPS 2002 Workshops, 2002.

[38] Subramani V, Kettimuthu R, Srinivasan S, Sadayappan P. Distributed Job Scheduling on Computational Grid Using Multiple Simultaneous Requests. In: the 11th IEEE International Symposium on High Performance Distributed Computing, IEEE Computer Society, 2002. 359-367.

[39] Casanova H, Zagorodnov D, Berman F, Legrand A. Heuristics for Scheduling Parameter Sweep Applications in Grid Environments. In: the 9th Heterogeneous Computing Workshop, IEEE Computer Society, 2000. 349-263.

[40] Ding Q, Chen G. A Benefit Function Mapping Heuristic for a Class of Meta-tasks in Grid Environment. In: the 1st IEEE/ACM International Symposium on Cluster Computing and the Grid, IEEE Computer Society, 2001. 654-659.

[41] Takefusa A, Casanova H, Matsuoka S, Berman F. A Study of Deadline Scheduling for Client-Server Systems on the Computational Grid. In: the 10th IEEE International Symposium on High Performance Distributed Computing, IEEE Computer Society, 2001. 406-415.

[42] Di Martino V, Mililotti M. Scheduling in a Grid Computing Environment Using Genetic Algorithms. In: the International Parallel and Distributed Processing Symposium, IEEE Computer Society, 2002. 235-239.

[43] Hamscher V, Schwiegelshohn U, Streit A, Uahyapour R. Evaluation of Job-Scheduling Strategies for Grid Computing. In: the 1st IEEE/ACM International Workshop on Grid Computing, LNCS, Vol. 1971, 2000. 191-202.

[44] Buyya R, Abramson D, Giddy J, Stockinger H. Economic Models for Resource Management and Scheduling in Grid Computing. In: Concurrency and Computation: Practice and Experience, Vol. 14 (13-15), 2002. 1507-1542.

[45] Wolski R, Plank J, Brevik J, Bryan T. Analyzing Market-Based Resource Allocation Strategies for the Computational Grid. In: The International Journal of High Performance Computing Applications, Vol. 15(3) 2001. 258-281.

[46] Buyya R, Murshed M. GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing. In: Concurrency and Computation: Practice and Experience (CCPE), Wiley Press, Vol. 14, Issue 13-15, 2002. 1175-1220.

[47] Nakai J. Pricing Computing Resources: Reading Between the Lines and Beyond. NAS Technical Report: NAS-01-010, NASA Ames Research Center, 2002.

CHAPTER 6

第6章 网格资源管理体系结构

资源管理和调度的目标是通过高效的管理和调度模式为多个相互独立的任务分配所需的计算资源，使系统的利用率最高。资源管理和任务调度方面的研究主要集中在调度策略和具体的调度算法。研究适用的调度策略与算法必须考虑底层网格基础设施的结构特性，因此，在进行调度算法研究之前，需要了解哪种体系结构更加适合现有的网格系统，采用何种资源管理模式能够更好地支撑后续研究的成果。本章针对现有资源管理模式不能很好地反映网格资源动态特征的问题，提出了基于对等网络的混合式网格资源管理体系结构。它是本书网格计算提出的一些模型和算法的基础。

6.1 网格体系结构

在对网格资源管理做相关研究之前，需要首先对网格的整体框架有一个全面的认识，即需要对网格体系结构有充分的认识。网格体系结构讨论了构造网格的技术，给出了网格的基本组成与功能，描述了网格各组成部分的关系以及它们集成的方法，刻画了支持网格有效运转的机制。

6.1.1 网格的层次结构

目前网格的结构是层次化的，如图6-1所示，自下而上的分层组件分别如下：

(1) 网格资源：包括网络上所有分布的、可访问的计算资源。仅仅实现了计算资源在物理上的连通。

(2) 网格中间件：包括一系列工具和协议软件。屏蔽网格资源的分布、异构特性，提供透明、一致的接口。

(3) 网格开发环境和工具：让开发人员开发不同的应用以及用户代理在全局资源中调度计算。

(4) 网格应用层：是用户需求的具体体现。提供网格入口(Grid Portals)技术。

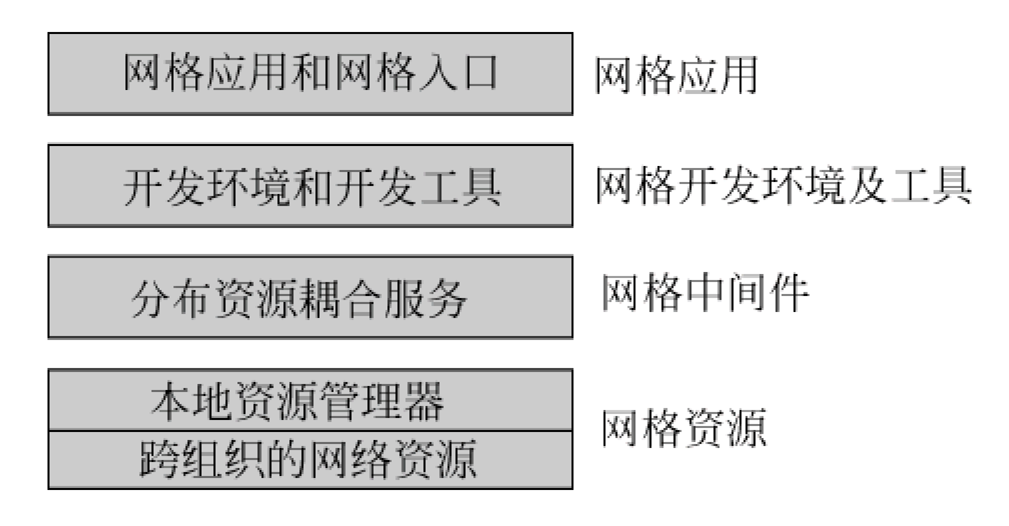


图 6-1 网格系统的组件模型

相应的网格协议体系结构也是分层次的，图 6-2 所示是面向协议的网格五层体系结构[1]，即 Globus 的体系结构。Globus 的协议分为五层：构造层、连接层、资源层、汇集层、应用层。上述五层，每层都有自己的服务、API 和 SDK，上层协议调用下层协议的服务。网格内的全局应用都通过协议提供的服务调用操作系统。

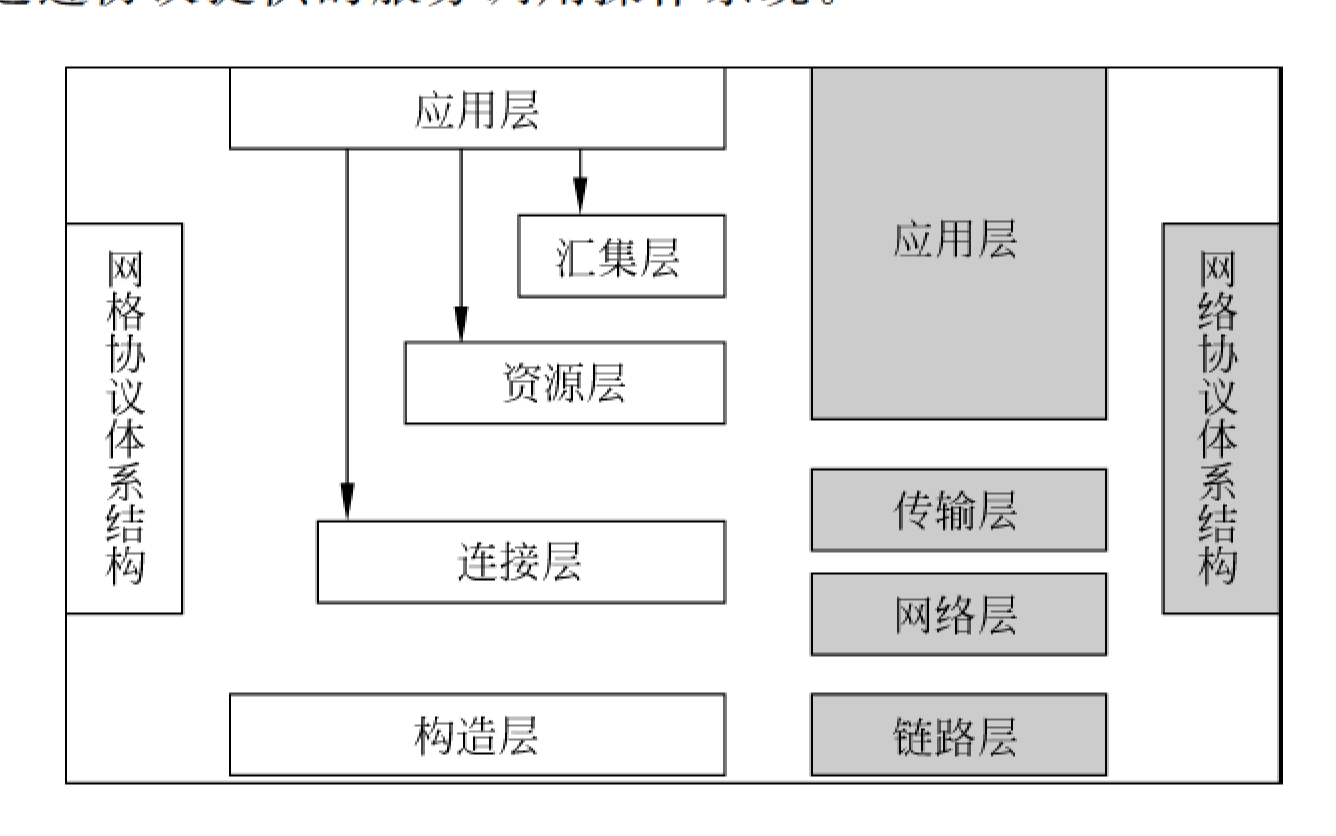


图 6-2 Globus 网格协议体系结构

(1) 构造层：功能是向上提供网格中可供共享的资源，它们是物理或逻辑实体。常用的资源包括处理能力、存储系统、目录、网格资源、分布式文件系统、分布式计算机池、计算机集群等。Toolkit 中相应组件负责侦测可用的软硬件资源的特性、当前负荷、状态等信息，并将其打包供上层协议调用。

(2) 连接层：是网格中网络事务处理通信与授权控制的核心协议。构造层提交的各种资源间的数据交换都在这一层的控制下实现。各资源间的授权验证、安全控制也在这里实现。在 Toolkit 中，相应组件采用基于公钥的网格安全基础协议(GSI)。GSI 提供一次登录、委托授权、局域安全方案整合、基于用户的信任关系等功能。资源间的数据交换通过传输、路由及名字解析实现。

(3) 资源层：作用是对单个资源实施控制，与可用资源进行安全握手、对资源做初始化、监测资源运行状况、统计与付费有关的资源使用数据。在 Toolkit 中有一系列组件用来实现资源注册、资源分配和资源监视。Toolkit 还在这一层定义了客户端的 C 语言和 Java 语言的 API 和 SDK。

(4) 汇集层：作用是将资源层提交的受控资源汇集在一起，供虚拟组织的应用程序共享、调用。为了对来自应用的共享进行管理和控制，汇集层提供目录服务、资源分配、日程安排、资源代理、资源监测诊断、网格启动、负荷控制、账户管理等多种功能。

(5) 应用层：是网格上用户的应用程序。应用程序通过各层的 API 调用相应的服务，再通过服务调用网格上的资源来完成任务。应用程序的开发涉及大量库函数。为便于网格应用程序的开发，需要构建支持网格计算的库函数。

面向协议的五层体系结构剖析了网格的内部构造，而面向服务的开放网格体系结构则是为了阐明网格外部功能特征[2]。同时，网格技术需要同现有的主流技术相融合，因此，美国的 Argonne 国家实验室、芝加哥大学、南加州大学和 IBM 公司共同提出了开放式网格服务体系结构(Open Grid Service Architecture，OGSA)。开放网格服务体系结构结合了现有的网格技术和 Web 服务技术，仍然采用 Globus 的五层结构，如图 6-3 所示。

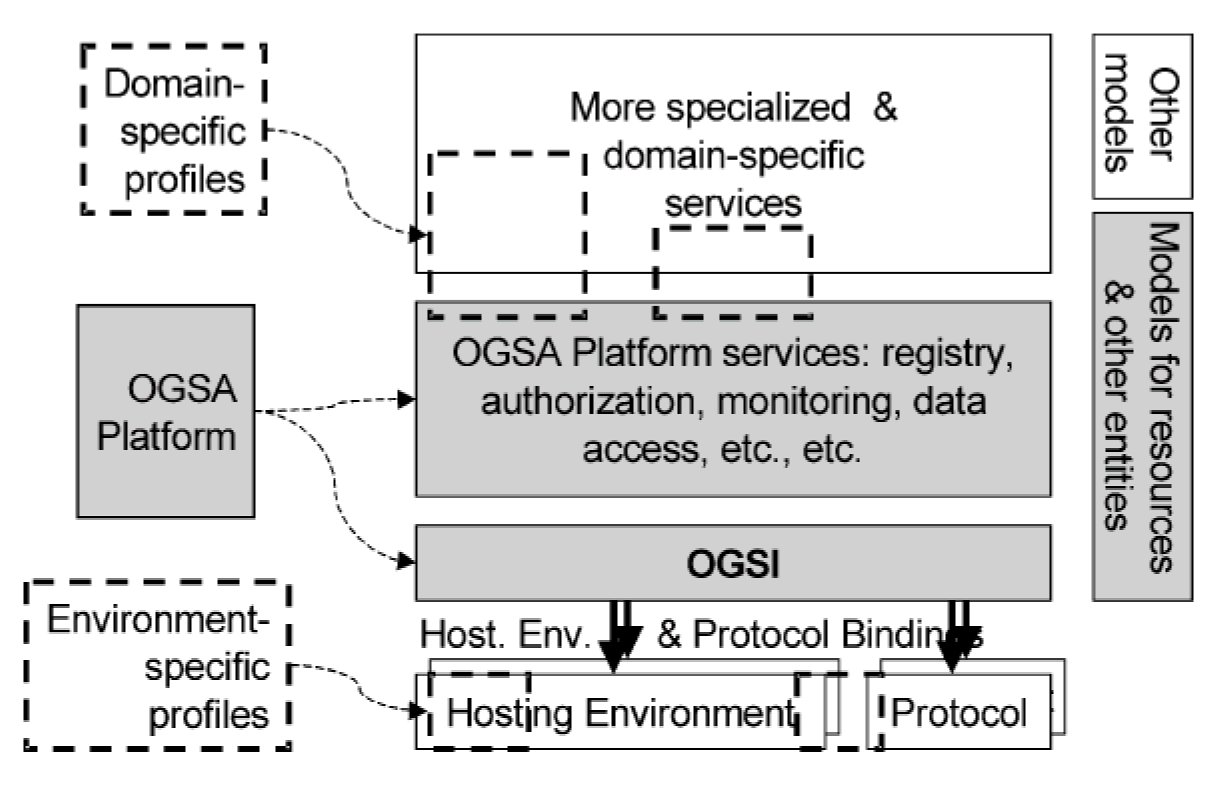


图 6-3　OGSA 体系结构

在网格计算中有一个十分重要的概念，即在动态的、多机构组成的虚拟组织中协同资源共享和问题解决。在这里，共享不仅停留在简单的文件交换层面上，更着重强调直接对计算机、软件、数据以及其他资源的直接访问，这种需求在工业、科学研究等很多领域都会遇到。同时，这种共享必须是高度可控的，需要在资源提供者和消费者之间详细地定义共享资源种类、共享权限以及共享的条件。虚拟组织就是基于这样的一些共享规则，由多个机构或个人组成的集合体。组成网格系统的资源是广域分散的，不再局限于单台计算机和小规模局域网范围内。网格系统的最终目标是将网络上的多种资源进行动态组合，为虚拟组织服务。

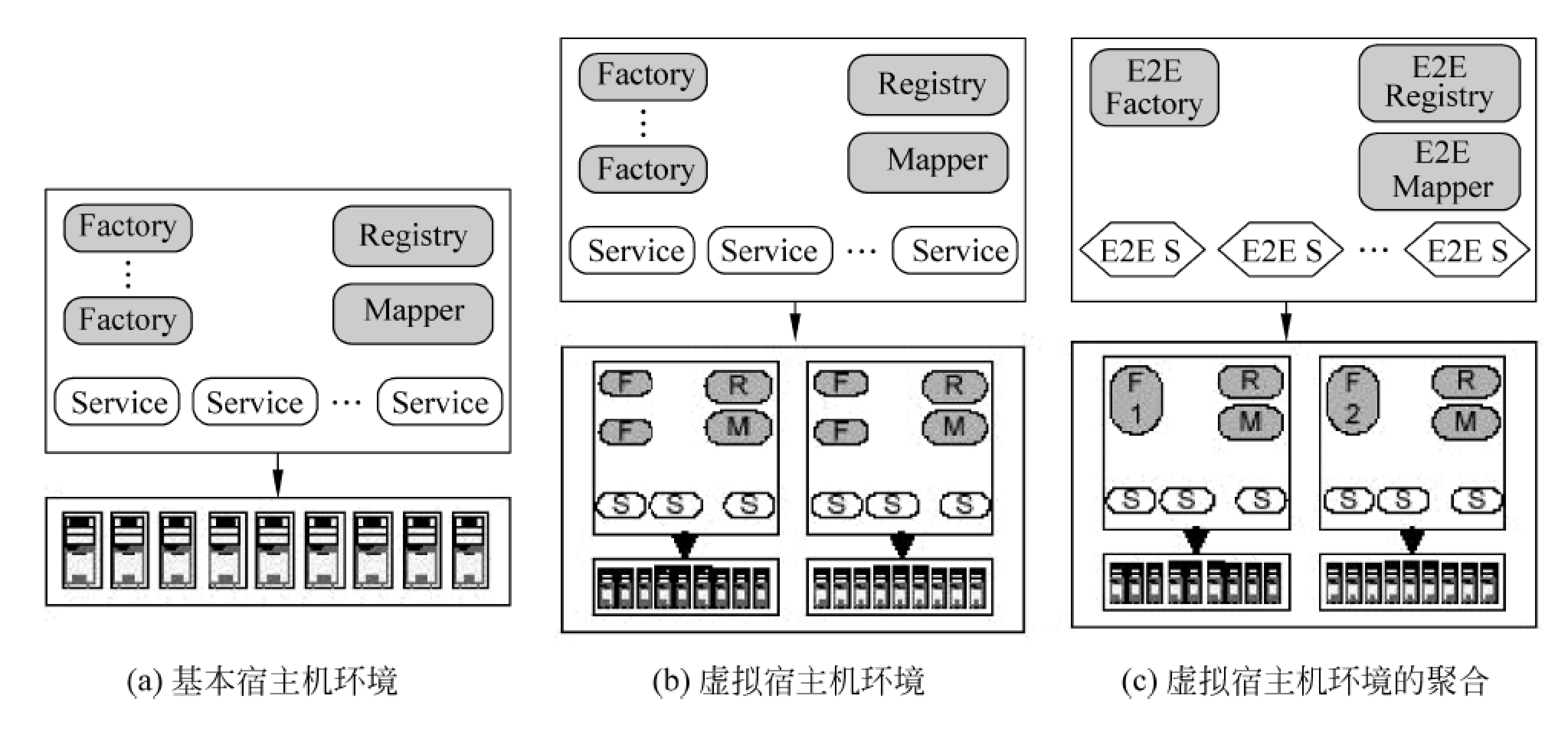
在 OGSA 中，基于服务导向和虚拟化，定义了一组 Grid Service 语义，与宿主机环境相结合来构建虚拟组织。

基本宿主机环境(Hosting Environment)如图 6-4(a)所示，一个简单的宿主机环境就是一个资源管理域内的资源集合，为服务管理提供本地设施。例如，一个 J2EE 应用服务器、Microsoft. NET 系统和 Linux 集群。在 OGSA 中，用户与环境的接口包括一个 Registry、一个或多个 Factory 和一个 Mapper 服务。每个 Factory 都被记录在 Registry 中，使得客户端能够发现可用的 Factory。当 Factory 接收到客户端创建 Grid Service 实例的请求时，该 Factory 激活宿主环境特定的功能创建一个新实例，为其分配一个句柄，并在 Registry 中注册该实例，并使得该句柄对 Mapper 服务生效。各种服务直接被映射到本地操作上。

虚拟宿主机环境如图 6-4(b)所示，在虚拟组织中的资源将跨越多个异构的、地理上分布的宿主机环境。尽管如此，通过使用与基本宿主机环境中相同的接口，客户端可以访问这种虚拟宿主机环境。在虚拟宿主机环境中，可以创建一个或多个高层 Factory，将创建请求委派给底层 Factory。同样，可以创建一个高层 Registry，它知晓高层 Factory 以及它们所创

建的服务实例,并熟悉特定于该虚拟组织的服务管理策略。这样,客户端可以使用虚拟组织的 Registry 查找 Factory,以及其他与该虚拟组织相关的服务实例,并使用 Registry 返回的句柄与这些服务实例直接联系。高层 Factory 和 Registry 都实现标注接口,因此从用户的角度来看,它们和其他 Factory 和 Registry 没有区别。

在虚拟宿主机环境基础上,还可以构建提供更为复杂、虚拟化、"聚合"或"端到端"服务的虚拟宿主机环境,如图 6-4(c)所示。在这种情况下,Registry 跟踪创建高层服务实例的 Factory,并发布相关信息。通过向底层 Factory 请求创建多个服务实例,并整合多个底层服务实例的行为从而实现单一高层服务实例。



(a) 基本宿主机环境　　(b) 虚拟宿主机环境　　(c) 虚拟宿主机环境的聚合

图 6-4　虚拟组织的构造

以上几个层次上虚拟组织实例,阐明了在 OGSA 框架下,如何在多个层次上构造虚拟组织,实现跨机构边界的分布资源整合以及在商业 IT 基础设施内部分布资源的整合。基于向本地平台资源和 API 的映射而实现的 Grid Service,可以无缝整合成高度的 Grid Service,多个虚拟组织中服务集合可以映射到相同的地层物理资源上,而且在同一层逻辑上不同的服务也可以共享底层的物理资源系统。

6.1.2　计算经济网格体系结构

网格系统中的资源供需关系,与现实世界中的商品经济模型存在类比关系:资源的提供者相当于商品制造商,为用户提供计算资源,并从中获利;资源使用者相当于商品购买者,为了满足自己的资源需求支付一定的费用。它们都是由利益驱动的,为了获得最大利益而制订策略。据此,Rajkumar Buyya 提出了经济网格及计算经济网格体系结构模型[3],如图 6-5 所示。

图 6-5 的模型侧重于研究网格资源的分配方式,主要包括以下四部分:

(1) 网格服务供应商(Grid Service Providers):包括地理上分布的所有互联资源和网格中本地资源管理系统,主要是拥有资源的服务供应商(相当于网格结点)对资源进行合理的分配,以满足交易服务器和资源预留的要求。同时服务供应商按照经济模型制定价格策略和交易原则,以求获得最大利益。交易服务器是服务供应商本身的代理程序,遵循价格策略与用户协商资源使用价格,指导结账系统记录资源消耗情况。网格服务供应商和网格用

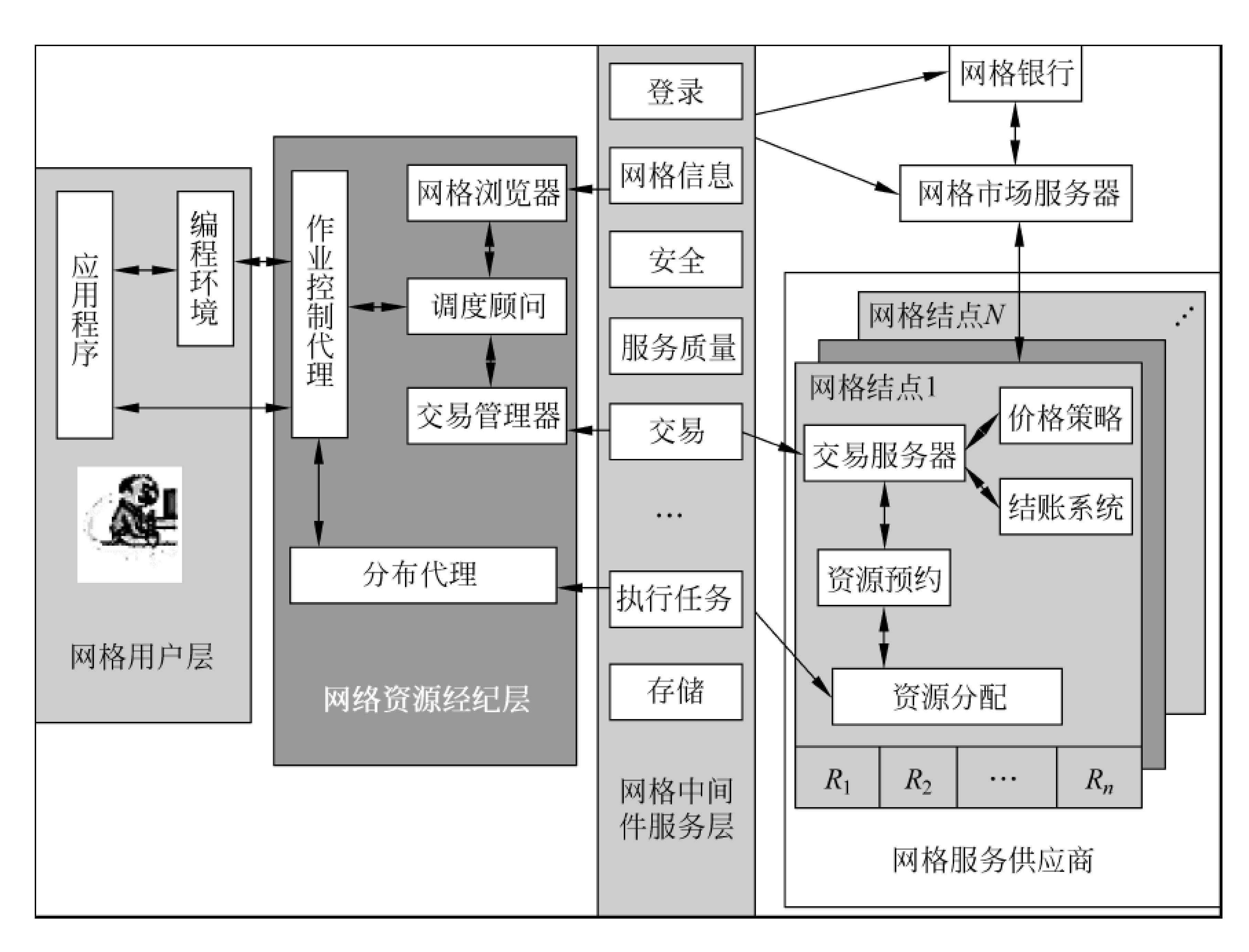


图 6-5　计算经济网格体系结构模型

户通过网格银行和网格市场服务器进行结算。

(2) 网格中间件服务层(Grid Middleware Services)：利用网格中间件实现了构造经济网格模型所需的各种服务，其中包括登录、安全和服务质量控制，提供必要的网格信息供网格浏览器查找所需资源，通过交易中间件连接交易服务器和交易管理器，执行任务模块完成资源的正确分配，存储中间件存储各种必要信息等。

(3) 网格资源经纪层(Grid Resource Broker)：负责资源发现、资源选择、软硬件资源的绑定、计算初始化、单一资源映像等功能。网格资源经纪层由作业控制代理、调度顾问、网格浏览器、交易管理器和分布代理五部分组成。作业控制代理作为总控负责监督程序的执行，连接用户和其余四部分。调度顾问负责资源发现和资源选择。交易管理器是在调度顾问的资源选择算法指导下，评估资源的访问开销，为调度顾问提供资源选择依据。分布代理负责在选定的资源上激活任务，并周期性地向作业控制代理更新任务执行状态。

(4) 网格用户层(Grid Consumer)：用户对网格系统提出自己的要求，包括所需资源和交易所必须的信息，如价格、时间限制、偏好等。用户可使用网格应用程序，直接向网格资源经纪层提出要求。或者用户利用网格语言编程环境，编写自己的网格程序提交给网格资源经纪层。用户可以通过 Web 在远程资源上提交数据和收集结果。

6.2　基于超级结点对等网络的网格资源管理体系结构研究

我们知道，未来网格系统具有对等网络的特点，资源的动态性和异构性是网格资源的固有特征，自治性也是其重要特征之一，因此，网格资源管理需要在已有的分布计算资源管理基础上，融合对等计算技术，作更进一步的研究。本章将对等网络技术引入网格计算，利用

对等网络的优点，设计了混合式的网格资源管理体系结构，本节的内容是本书后续网络计算研究的基础。

6.2.1 对等网络组织类型

目前对等P2P计算技术被广泛应用于各个领域，如信息资源共享、普及计算、协同工作、实时通信技术、信息检索技术、广域网络存储系统等。对等计算技术的出现使得充分利用互联网中所蕴含的潜在计算资源成为可能。对等网络的每个结点既是客户机，又是服务器，还是路由器。P2P系统中的各个结点因为互为服务而共存，而不是依赖与特定的集中式机制。而且，各个结点可以直接交互并可能随时离开对等网络。与传统系统相比，对等网络具有以下特征：

(1) 结点之间通过直接交互共享资源。

(2) 资源分布在各个结点中，而不是集中在一个服务器中进行管理。

(3) 结点具有动态性与即席性。

(4) 纯粹的P2P系统没有任何集中控制机制。系统的各结点运行的P2P系统软件功能相同，各结点之间交互对称。

和传统Client-Server网络相比，对等网络具有如下优势：

(1) 整体性能高。对等网络中，没有服务器、客户机之分，可以充分利用各参与者提供的共享资源，实现网络整体性能的最大化。

(2) 可扩展性好。在对等网络中，系统自动适应参与者的加入和退出。随着参与者的增加，网络规模越来越大，网络中可共享的资源越来越丰富。

(3) 容错性好。当网络中参与者较多时，资源冗余成为必然，从而保证了资源的可得性和抗毁性；冗余还可以避免“单点失效”问题。对等网络中参与者具有分散性等特性，这进一步提高了系统资源的可靠性。

(4) 信息内容更丰富。各个网络参与者都可能是信息提供者，随着网络规模的扩大，信息必将越来越丰富。当网络增长的时候，共享信息的数量和范围都将随之增长。在一个开放网络环境下，P2P网络能够很快积累相当丰富的信息。

(5) 负载均衡。对等网络环境下可以根据策略灵活分布信息。负载均衡模块可以监控各种信息的流量和请求率，然后重新分布这些信息以减轻单个结点的负载。通过这种负载平衡策略可以提供分布式Cache可以实现的功能，但是更简单而且代价小。

现有的对等网络可按结构特性和集中化程度分类[4]。按结构特性可分为非结构化对等网络和结构化对等网络。在非结构化对等网络中，例如Gnutella[5,6]，对等结点连接任意其他对等结点构成对等网络，数据放置与对等网络拓扑无关，网络拓扑是ad hoc的。Chord[7]、Tapestry[8]、Pastry[9]和CAN[10]则属于结构化对等网络。这类网络提供了文件ID和文件存储位置之间的映射关系，从而保证有效路由，并保证最终找到目标结点。

对等网络按集中化程度可分为纯对等网络和混合对等网络。纯对等网络，例如Freenet[11]，采用完全分布的对等网络查找过程，网络中每个结点完全对等。结点在各方面，如查询、下载和路由等，有相同的角色和责任。混合对等网络，例如Napster[12]则采用集中式资源查找方法，网络中结点责任不完全对等，处理能力强的结点作为其他结点的目录服务器。混合网络的特点是集中式查找，分布式下载。

类似 Morpheus[13]的超级结点对等网络系统是目前最流行的文件共享系统之一。超级结点对等网络 Kazza[14]是纯对等网络与混合对等网络之间的一种折中。在超级结点对等网络中,结点按地理邻近性划分为聚类。每一聚类内有一个或多个超级结点。超级结点作为聚类内其他结点的目录服务器。聚类内的其他结点称为客户结点。客户结点将查找请求提交给超级结点并从超级结点接收查找结果。客户结点与超级结点之间的关系类似于混合对等网络中对等结点与集中式目录服务器之间的关系。超级结点之间采用了纯对等网络的联系方式,且超级结点接收客户结点提交的查找请求,并代表客户结点路由该查找请求。由上可知,超级结点在路由方面是对等的,而客户结点在下载时是对等的。超级结点对等网络就是由超级结点及所属客户结点所组成的对等网络。

按超级结点的组织方式,超级结点对等网络可分为结构化超级结点对等网络及其非结构化超级结点对等网络。在结构化超级结点对等网络中,超级结点采用结构化对等网络的组织方式,例如 Chord、Tapestry、Pastry、CAN 等;而在非结构化超级结点对等网络中,超级结点之间采用非结构化超级对等网络的组织方式,例如 Freenet, Gnutella 等。参考文献[15]中提出了一种非结构化超级对等网络的结构。该网络在聚类内部采用 Napster 集中式的资源查找方法,在超级结点之间采用 Gnutella 扩散查找方法。参考文献[16,17]中提出了一种结构化超级结点对等网络,其中的超级结点对等网络分别采用 CAN 和 Tapestry 组织超级结点。

6.2.2　各种对等网络分析

各类对等网络都各有优缺点[18]。混合对等网络采用集中式目录服务器方式查找信息资源。这种查找方式的优点是查找效率较高且易于管理,但集中式目录服务器容易成为单点故障和性能瓶颈。

非结构化对等网络的搜索过程类似于随机搜索过程。由于数据的放置与网络拓扑无关,我们很难期望非结构化对等网络获得较优的查找路线。对等网络的消息开销较大,如 Gnutella 需采用扩散的方式转发查找请求,并通过 TTL 控制扩散的深度。非结构化对等网络的优点是对网络的动态性适应能力强;缺点是当 TTL 过小时,很难找到目标文件,而当 TTL 过大时,由于消息数量随着路由的深入呈几何指数增加,又会在网络上生成过多的查找消息。

结构化对等网络的数据放置与网络拓扑相关。结构化对等网络能保证路由有效进行。但在非常动态的网络环境下,结点的频繁加入与退出需要较大的维护开销,并且当结点退出以后,路由表没有及时重建也会导致查找失败。

对等网络由于存在上述不足,很难扩展为大规模系统。超级结点对等网络从某种程度上克服上述对等网络所存在的缺陷。由于超级结点作为客户结点的目录服务器,因此超级对等网络查找效率高于纯对等网络。与此同时,由于系统中有比较多的超级结点,超级结点间能有效地平衡负载,从而在一定程度上减轻了单点故障和性能瓶颈问题。与现有的对等网络相比,超级结点对等网络有潜在的优势。

6.2.3　基于超级结点对等网络的网格资源管理体系结构

基于对各种对等网络组织类型的分析,我们认为,由于超级结点对等网络的超级结点间

能有效地平衡负载,从而在一定程度上减轻了单点故障和性能瓶颈问题。同时因为超级结点是客户结点的目录服务器,其资源发现和资源查找的效率比较高,因此,与现有的对等网络相比,超级结点对等网络具有潜在的优势。基于此,我们将利用类似超级结点对等网络的体系结构来进行网格资源的管理和调度。

Iamnitchi[19]提出:在大规模、动态、异构的网格环境下,可以不失一般性地假设存在一个或多个超级结点,每个超级结点上都记录有一组资源信息。超级结点之间的关系是对等的,它们在地理上分布、可动态加入和退出。这些超级结点之间的拓扑是可以动态变化的,将它们之间的连接拓扑看成是一个覆盖网络(Overlay Network),其拓扑结构的选择对网格资源管理的性能有很大的影响,好的 Overlay Topology 将有助于提高资源管理性能。

本节将网格资源提供者组织在一个 P2P Overlay Network 中,网格资源提供者可以是一台 PC 机,或者是具有很多处理结点的集群,也可以是一台多功能设备,甚至是移动设备,例如 PDA、手机等。这些不同种类的网格资源提供者根据自己的地理位置、兴趣、共享方式等组成不同的虚拟组织 Virtual Organizations,每个 VO 中都存在一个超级结点来管理 VO 内部以及 VO 之间的通信。这些虚拟组织之间是相互对等的,通过超级结点进行相互通信,维护 P2P Overlay Network 的拓扑结构。超级结点可以是一个真正的结点,例如一台超级计算机的全局管理结点,也可以是一段驻留在某个结点上的虚拟程序。特别地,可以将集群看成是多个结点的集合,其全局管理结点为超级结点(Super Node),其他计算结点为客户结点(Client Node);将 PC 机看成是没有客户结点的超级结点;或者将集群和 PC 机看成是同一个 VO 内部的客户结点,而该 VO 超级结点是一个进程。该进程可以随机选择其驻留的结点,集群或者 PC 机,考虑到存储空间和处理能力的因素,以及负载均衡的影响,在构建 VO 时往往选择在处理能力强的结点上驻留此进程,例如超级计算机或者集群,而不选择 PC 机或者移动设备。由这些虚拟组织构成的 Overlay Network 具有类似超级结点对等网络的拓扑结构。超级结点之间采用纯对等网络的连接方式。图 6-7 是这种连接方式的一个简单示意图。

由于每一个资源提供者都是这个 Overlay Network 中的一个对等结点。如果一个结点失效了,Overlay 网络上的其他结点可以代替此结点完成任务,以此来确保网格系统的鲁棒性和容错性。

图 6-6 所示的每个超级结点都拥有一些共享内容:其所辖域内的 Client Node 的共享内容,如可利用的 CPU 数量、存储空间、工作负载信息、结点操作系统版本等;其他对等的超级结点的索引信息表的拷贝,各超级结点利用该索引表来维护 Overlay Network 拓扑结构的路由信息等。超级结点通过查找所拥有的索引表(包括其他结点的索引表拷贝)来响应资源查询请求。

另外,每个超级结点上还有一个本地调度器,这个本地调度器其实就是一个进程,负责维护超级结点的本地共享信息,维护与其他结点连接的索引表,同时负责客户结点间的本地资源管理和调度。

由以上分析可知,超级结点在路由方面是对等的,而客户结点在处理网格并行任务时是对等的。

网格资源管理 Overlay Network 的组织方式时有两种选择:非结构化组织方式和基于分布式哈希表(DHT)的结构化组织方式。在选择图 6-6 所示的 Overlay Network 的组织方

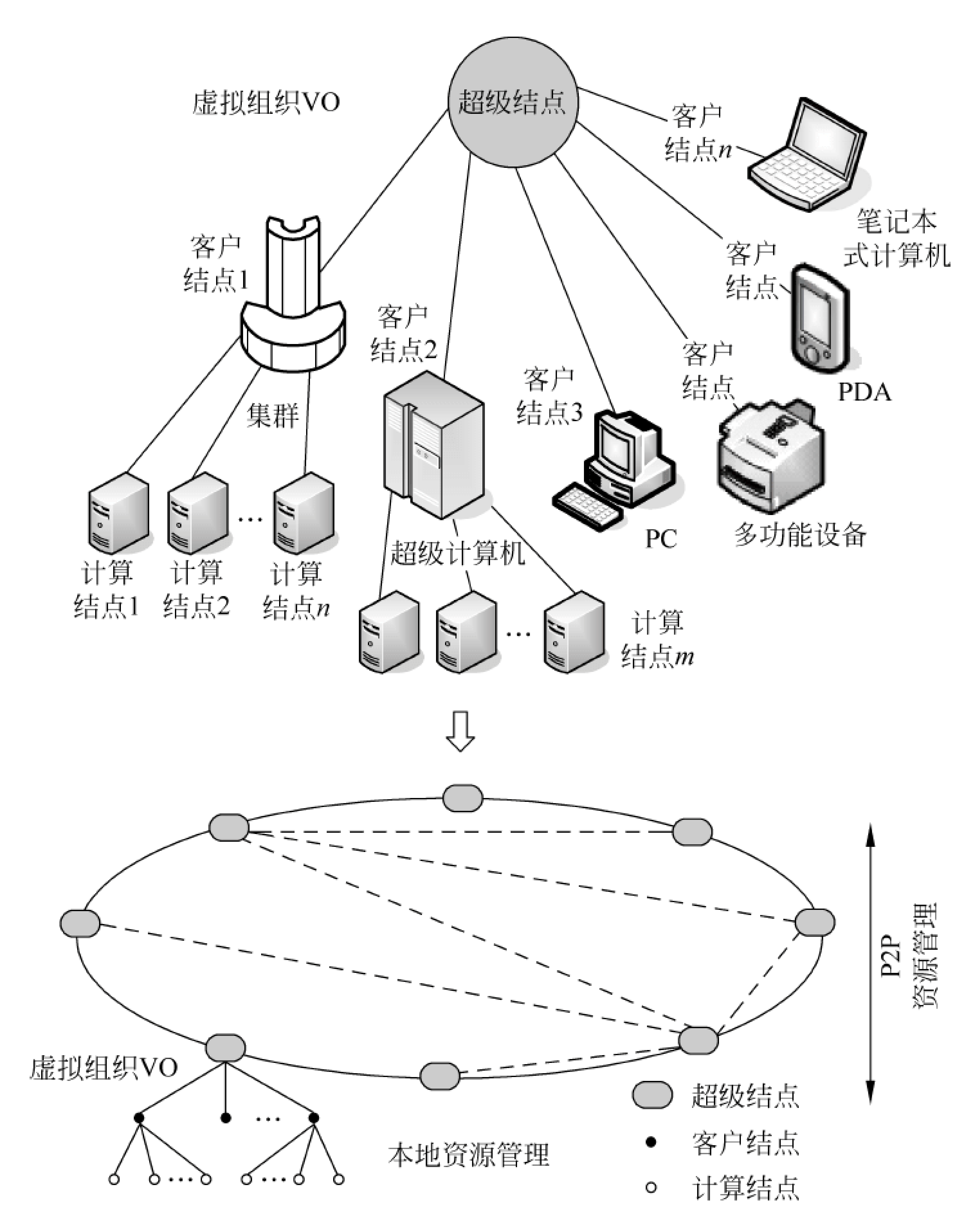


图 6-6 基于超级结点对等网络的网格资源管理体系结构

式时,需要考虑两个方面的问题:

(1) 超级结点之间的对等网络的组织方式,有两种选择:结构化的组织方式和非结构化的组织方式。

(2) 虚拟组织内部的客户结点之间的组织方式,也有两种选择:集中式(如 Napster)组织方式和分布式(如 Gnutella)组织方式。

网格资源调度过程中的资源发现和资源查找(Mapping 过程)与非结构化对等文件搜索有明显的不同,区别在于:

(1) 网格资源管理和调度过程中,在 Overlay 中查找的是可用资源的信息,不是文件,这些资源的信息实时性高,是动态变化的,需要及时更新,在其他结点做备份的意义不大。因此,和分布式哈希表维护文件存储必须考虑用缓存文件来提高查找性能不同,网格资源管理 Overlay 中不需要为实时的可用资源信息进行大量的缓存备份。在网格系统中,每个超级结点仅拥有其所辖虚拟组织内部的客户结点的本地共享信息以及其他超级结点的索引信息表的拷贝。

(2) 网格资源管理和调度过程中,可利用资源信息的频繁更新会带来极大的路由和存储空间开销,资源管理和调度的效果以及整个网络系统的稳定性必将受到影响。

在进行网格资源调度时，超级结点通过查找所拥有的索引表(包括其他超级结点的索引表拷贝)来响应用户任务对网格资源的请求。如何选择有效的路由转发机制对于调度算法的性能将产生很大的影响。在非结构化对等网络中经常采用泛洪(Flooding)的路由转发机制，但是 Flooding 方式往往会造成过重的负载和过多的网络流量。另外，随机转发(Random Walk)以及在 Random Walk 上演化出来的路由转发机制也是可供考虑和选择的路由转发机制。

使用何种方式组织 Overlay Network，采用何种路由机制进行信息转发，在接下来的几章中，针对具体的调度机制和算法，本节将做详细的分析和描述。

本节设计的网格资源管理体系结构和现有的网格体系结构相比，具有以下几方面的优点：

(1) Globus 和 OGSA 的体系结构是层次式的，其面向协议的五层结构在实现的过程中集合了集中式和分布式资源调度模式的优点，能够在一定程度上消除两种模型的缺点，同时在一定程度上解决资源的自治性、异构性以及扩展性等问题。但是在处理动态变化较快的资源环境时，Globus 和 OGSA 显现了一定的局限性，因为资源管理还是需要通过访问目录服务器来完成，这无疑会造成信息的过时和失效。而本节提出的基于超级结点对等网络的网格管理体系结构，由于虚拟组织之间采用了对等网络技术，在处理结点的动态接入和退出时，只需要维护部分邻居结点的路由索引信息表，或者采用 Flooding 的方式在一定范围内进行更新，系统维护量小，从而可以进行有效的动态管理和调度。

(2) 层次式管理模式的网络拓扑结构一旦确定下来，很难改变，无法自动地根据网格结点的动态加入和退出选择合适的连接拓扑，从而不利于提高资源管理的性能。而本节设计的超级结点对等网络，超级结点之间的拓扑结构是可以动态选择和变化的。在资源提供者发生变化时，可以根据网格系统的实际情况，选择不同的 Overlay Network 的拓扑组织结构，从而提高网格资源管理的效果。

6.2.4 P2P、网格、基于超级结点对等网络的网格比较

对结合本节对网格系统和对等网络的分析，以及基于超级结点对等网络的网格系统的设计，表 6-1 对 P2P 系统、网格系统和基于超级结点的对等网路系统做了比较，指出了三者之间的不同。

表 6-1 P2P、网格和基于超级结点对等网络的网格比较

	P2P	网　格	基于超级结点对等网络的网格
应用领域	通用/商用计算	全局问题求解环境	基于大型机、PC 机的各种网格计算
参与者	志愿者可随时加入或退出	预先确定，需注册	超级结点需预先确定，计算结点可以随时加入或者退出
可靠性	低：存在不可信的 Peer	高	动态可控
安全性	低	高	基于应用层的安全
可控性	不可控	集中控制	基于策略的部分可控
结点角色	Peer：可同时为服务器/客户端/路由器	Grid：服务器	Grid Peer：超级结点既是服务器又是客户端和路由器；计算结点为客户端

	P2P	网格	基于超级结点对等网络的网格
网络特征	各种速度的网络	静态高速	同时具有 P2P 的灵活性和网格的安全性、可调度性
资源管理	无调度器的资源管理	静态的中央注册中心	分布式的服务注册中心
资源调度	基于资源发现的调度	集中式的作业调度器	基于资源发现和网格服务发现 分布式和集中式作业调度器

6.3　基于超级结点对等网络的网格拓扑描述方法

理想的 Overlay Network 拓扑结构描述方法应该能够准确表示网格的物理连接信息、反映 IP 层和 Overlay 层间的对应关系，还应该能够简单、高效地刻画可供使用的网格资源信息，以易于资源调度算法发掘资源信息和任务请求二者之间的关系，从而进行高效的网格应用程序调度。

6.3.1　Overlay Network 拓扑的有向图描述

使用有向图（Directed Graph）来描述基于超级结点对等网络的网格系统的 Overlay network 拓扑结构，可以准确地反映网格系统的拓扑连接方式以及资源提供者的可利用资源信息。在用来表示网格 Overlay Network 拓扑的有向图中，图上的结点对应着网格资源提供者，边代表各结点之间的连接。该方法定义如下：

定义 6-1　结点的含义。使用有向图 G 表示网格资源管理 Overlay Network 拓扑。图 G 中结点的数量和结点之间的路由连接关系是动态变化的。结点个数为 n。存在两种类型的结点：超级结点和客户结点。图 G 中的超级结点在路由上是对等的，包含客户结点在内的图上所有结点在计算上是对等的。

定义 6-2　结点的功能。所有的结点都是信息结点。每个结点都包含一个路由表和信息表，路由表记录其他结点的路由信息，信息表用来存储本地资源信息。客户结点将本地资源信息和位置信息提供给所属的超级结点，每个超级结点拥有它所辖虚拟组织域内的所有客户结点的路由信息和资源信息备份。超级结点不使用其他的超级结点进行信息缓存备份，即每个超级结点的信息表仅对本地客户结点信息进行维护。

定义 6-3　边的含义。如果结点 X 有一条指向结点 Y 的有向边，则表示 X 有指向 Y 的路由表项。

定义 6-4　结点路由表和信息表的更新。当网格中有超级结点加入或者离开时，只需将它维护的路由信息表发布给其邻居超级结点；如果有客户结点的加入或离开时，客户结点将其动作消息发布给所属的超级结点，超级结点对路由表进行修改，同时，此超级结点通知其邻居超级结点进行信息更新。当客户结点更新资源信息时，只需更新自己的消息表中的信息，然后将新信息发送给其超级结点即可。

该方法还定义了三类有向边：非转发式索引连接（Non-Forwarding Index Link，NIL）、转发式搜索连接（Forwarding Index Link，FSL）和非转发式搜索连接（Non-Forwarding Search Link，NSL）。

(1) 非转发式索引连接用单虚线箭头表示。$X \dashrightarrow Y$ 表示在资源调度过程中,结点 X 向结点 Y 发送自身的索引信息更新,包含索引项的增加、删除和修改,结点 Y 及时更新自己维护的结点 X 的索引,但是 Y 不再进一步转发该索引通知。

(2) 转发式搜索连接用双实线箭头表示。$X \Rightarrow Y$ 表示在网格的资源调度过程中,结点 X 向结点 Y 发送资源请求,结点 Y 处理这个请求,并进一步通过由 Y 出发的 FSL 边转发查询请求。

(3) 非转发式搜索连接用单实线箭头表示。$X \rightarrow Y$ 表示在资源调度过程中,结点 X 向结点 Y 发送资源请求,结点 Y 处理这个请求,但是结点 Y 不再进一步转发该资源请求。

在使用有向图表示网格 Overlay Network 拓扑时,Overlay Network 中的客户结点使用一条 NIL 和其所属的超级结点连接;超级结点使用 NSL 和其所辖虚拟组织内部每个客户结点相连;而超级结点之间则采用 FSL 互连。

基于以上定义,网格 Overlay Network 可以被表述为一个特殊的有向图,该图为一个二元组(V,E),还有两个权值函数,w_v 和 w_e。(V,E)决定了网格的拓扑结构。其中,V 是网格结点集合,图中的每个结点唯一对应着网格中的一个结点,$|V|=n$,即结点个数;E 是网络连接的集合,是图中连接边的集合,$|E|=m$,即图中边的条数。

$E(V,E)$是 1-graph,即 $E \in V \times V$。模型中不存在自环,即不存在指向结点自身的直接边。对于边 E,用 $i(e)$表示它的起点,$t(e)$表示它的终点。

两个权值函数 w_v 和 w_e,分别对应结点的权值和边的权值。$w_v: V \rightarrow R$ 和 $w_e: E \rightarrow R$,R 代表正有理数集。结点 v 的权值是结点可以提供的计算资源的量化表示;边 e 的权值表示从边 E 起点 $i(e)$到终点 $t(e)$花费的时间,例如,$w_v(X)=m$,含义为网格结点 X 当前可提供的计算资源数量,用系统的 benchmark 值量化表示 m 个计算单元,此 benchmark 值由不同的网格系统在初始化时自行规定。$w_e(L)=n$,含义为从边 L 的起点到终点的时间耗费为 n(单位为 μs),此路由单位的值可以通过基准测试结合程序中的操作数等信息来估计。结点 X 的邻居结点集合表示为 neighbor(X)。

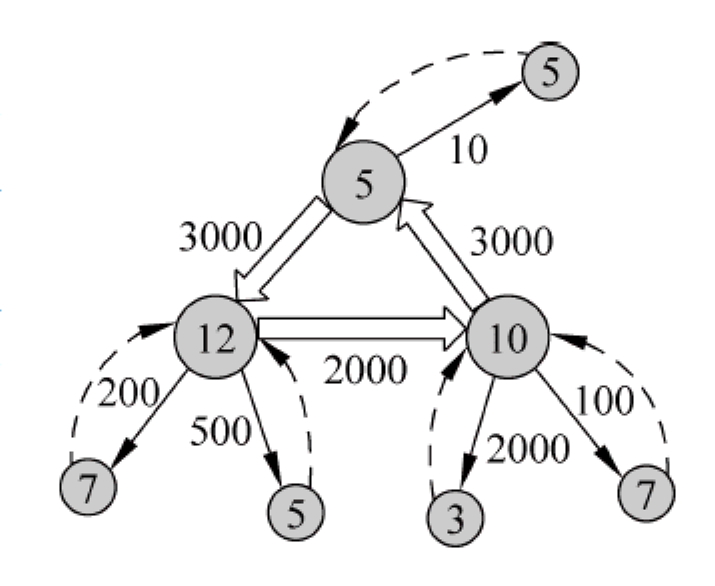


图 6-7 有向图模型描述网格资源覆盖网

图 6-7 所示为该有向图的可视化形式,所有的边连接用不同的箭头表示,表示结点的权值标记在结点内,边的权值标记在边的旁边,如果权值为∞,则不标记。

6.3.2 相关工作比较

由 Brain F. Cooper 和 Hector Garcia-Molina 提出的用于研究对等网络中的资源查找的 SIL(Search/Index Link)模型[20~22]也使用有向图来描述网络拓扑,图的顶点对应网络的结点,边代表网络中结点之间的连接。在 SIL 模型的有向图中,存在四种类型的有向边:转发式搜索连接、非转发式搜索连接、转发式索引连接(Forwarding Index Link,FIL)以及非转发式索引连接。和本书的有向图表示方法的定义相比,SIL 模型多了转发式索引连接 FIL 的边。

本书之所以仅定义三种类型的边来表示结点之间的连接,是由网格资源管理和调度的自身特点决定的。

首先，网格资源管理 Overlay Network 和 SIL 表示的查找网络存在不同。具有超级结点的查找网络包括两种行为：一种是查询，一种是索引更新。查询往往由客户结点发起，通过超级结点之间的连接将查询请求在整个 Overlay 网络中传播。因此，客户结点使用 FSL 和超级结点连接，超级结点间也使用 FSL 互连。索引更新也是由客户结点发起，当一个客户结点加入或者退出，或者其状态发生变化时，通过一条 NIL 通知其超级结点即可。

网格资源管理也包括两种行为：一种是资源调度；一种是结点索引更新，包括结点的加入、退出和实时信息更新。网格资源管理过程中的索引更新和查找网络的索引更新是类似的：客户结点发生状态改变，只需通过一条和超级结点相连的 NIL 连接，将新的状态传递给超级结点进行更新；同时，网格资源调度和分布式文件存储不同，不需要对结点的实时信息进行缓存备份，因为动态的信息更新使得路由和存储的开销太大。定义 6-2 和定义 6-4 规定，超级结点在本地维护其客户结点的资源信息，不需要使用其他结点进行备份，信息更新时客户结点仅需要通知其超级结点，而超级结点不需要向其他邻居超级结点转发该消息。此动作是单向的。所以，只需要使用一条由客户结点指向超级结点的 NIL 即可表示网格资源调度 Overlay 网络的索引连接。

其次，网格中的资源调度过程却是查找网络中查询行为的逆过程。本章讨论的广域资源调度在由超级结点组成的对等网络中进行，客户结点不参与广域资源调度过程。所以，在资源调度过程中，超级结点首先接收到用户任务对可用资源的请求，由超级结点将此请求转发到客户结点，如果在某个虚拟组织内部不能找到合适的资源时，超级结点再将此请求沿着由其出发的 FSL 连接转发给其他虚拟组织。而在查找网络中，查询请求首先由客户结点转发到超级结点，然后再在超级结点间转发。因此，在本章的有向图表示方法中，超级结点使用指向客户结点的 NSL 和其所在虚拟组织内部每个客户结点连接，而超级结点之间则采用 FSL 互连，表示方向和 SIL 的方向相反。

6.4　本章小结

本章在分析了现有网格资源管理体系结构、各种对等网络组织结构的基础上，提出了基于对等网络的网格资源管理体系结构，此结构采用类似超级结点对等网络的结构进行网格资源的管理，从而更好地描述网格资源的动态性，使网格系统具有很好的鲁棒性和容错性，消除单点失败和性能瓶颈的问题。进一步，本章论述了使用有向图表示网格 Overylay Network 拓扑结构的方法，给出了有向图的结点和边的详细定义。对如何进行 Overlay 组织形式和路由转发机制进行了讨论。本章的工作是以后各章提出的算法的研究基础。

参考文献

[1] Foster I，Kesselman C，Tuecke S. The Anatomy of the Grid：Enabling Scalable Virtual Organizations. In：International Journal of Supercomputer Applications，Vol. 15(3)，2001. 200-222.

[2] Foster I，Kesselman C，Nick J，Tuecke S. The Physiology of the Grid：An Open Grid Services Architecture for Distributed Systems Integration. In：Open Grid Service Infrastructure WG，Global Grid Forum，2002.

[3] Buyya R. Economic-based Distributed Resource Management and Scheduling for Grid Computing.

[Ph. D dissertation], School of Computer Science and Software Engineering, Monash University, Australia, 2002.

[4] Dingledine R, Freedman M J, Molnar D. The Free Haven Project: Distributed Anonymous Storage Service. In: the Workshop on Design Issues in Anonymity and Unobservability. 2000. 67-95.

[5] Gnutella. http://gnutella. wego. com.

[6] Stokes M. Gnutella2 Specifications Part One: http://www. gnutella2. com/gnutella2 search. htm.

[7] Stoica I, Morris R, Karger D, Kaashoek M F, Balakrishnan H. Chord: A scalable peer-to-peer lookup service for Internet applications. In: SIGCOMM 2001, ACM, 2001. 149-160.

[8] Zhao B, Kubiatowicz K, and Joseph A. Tapestry: An infrastructure for fault-resilient wide-area location and routing. Technical Report UCB//CSD-01-1141, University of California at Berkeley, April 2001.

[9] Rowstron A, Druschel P. Pastry: Scalable, Distributed Object Location and Routing for Large-scale Peer-to-Peer Systems. Lecture Notes in Computer Science, Vol. 2218, 2001. 329-350.

[10] Ratnasamy S, Francis P, Handley M, Karp R, Schenker S. A Scalable Content-Addressable Network. In: SIGCOMM 2001, ACM, 2001. 161-172.

[11] Clarke I, Sandberg O, Wiley B, Hon T. Freenet: A Distributed Anonymous Information Storage and Retrieval System. In: the ICSI Workshop on Design Issues in Anonymity and Unobservability, 2000.

[12] Document Object Model Level 2 Specification Napster. http://www. napster. com.

[13] Morpheus website. http://www. musiccity. com.

[14] Kazaa. http://www. kazaa. com.

[15] Yang B, Garcia Molina H. Designing a super-peer network. Technical report. http: //www-db. stanford. edu/peers.

[16] Zhao B, Duan Y, Huang L, Joseph A D, Kubiatowicz J D. Brocade: Landmark routing on overlay networks. In: the 1st International Workshop on Peer-to-Peer Systems (IPTPS 02), 2002.

[17] Kubiatowica J, Bindel D, Chen Y, Eaton P, Geels D, Gummadi R, Rhea S, Weatherspoon H, Weimer W, Wells C, Zhao B. OceaLSTore: An Architecture for Global-scale Persistent Storage. In: ACM ASPLOS, ACM, 2000.

[18] Qin L, Ratnasamy S, Shenker S. Can heterogeneity make gnutella scalable? In: the 1st International Workshop on Peer-to-Peer Systems (IPTPS2002), 2002.

[19] Iamnitchi A, Foster I, Nurmi D. A Peer-to-Peer Approach to Resource Discovery in Grid Environment. Technology Report, University of Chicago, 2002.

[20] Brian F C, Molina H G, ect. Studying Search Networks with SIL. In: the 2nd International Workshop on Peer-to-Peer Systems, 2003.

[21] Cooper Brian F, Hector Garcia-Molina. SIL: Modeling and measuring scalable peer-to-peer search networks. In: International Workshop on Database, Information Systems and Peer-to-Peer Computing, 2003.

[22] Cooper Brian F, Hector Garcia-Molina. Ad hoc, self-supervising peer-to-peer search networks, Technology Report, Department of Computer Science Stanford University, 2003.

CHAPTER 7

第7章 网格资源调度算法研究

7.1 基于树匹配的网格资源调度算法研究

影响网格资源调度算法性能的因素有：资源描述、任务请求描述、拓扑结构、消息的路由机制等。本节提出了基于树匹配的网格资源调度 nTreeMatch 算法。算法尤其适用于为解决任务并行度高的计算密集型科学应用而设计的专用计算网格。使用 DAG（有向无环图）描述网格任务，在调度过程中充分利用描述 Overlay Network 拓扑结构的有向图中的结点资源信息和路由信息，尽量使 Overlay 层上的路由跳数接近 IP 层上的路由跳数，降低 RDP。大量的实验证明算法能很好地进行网格资源的动态调度，具有良好的负载均衡效果。

7.1.1 网格资源信息描述方法

理想的网格资源信息描述方法应该在准确描述网格资源提供者的计算能力的同时，还能够利于网格资源调度器发掘网格资源提供者和网格任务之间的对应关系，此外，还应该能够反映资源提供者之间的相互关系，例如路由连接、从属关系等。

nTreeMatch 算法在选择网格资源信息描述方法时，采用了第6章提出的网格 Overlay Network 拓扑有向图表示方法。如定义 6-2 所述，该有向图中的每个结点都包含一个信息表和一个路由表。信息表记录了该网格结点当前的计算能力，如果结点为超级结点，则表示所辖域内所有客户结点的计算能力。路由表存储了该结点的位置信息，以及和其他结点的路由连接情况，客户结点仅维护和其超级结点的连接，而超级结点除了需要维护其所辖域内的链路信息外，还需要维护和其他超级结点间的路由信息。如果在生成此有向图时考虑网格结点 IP 层上的物理连接信息，生成含有最短路由耗费信息的路由表，那么在此基础上设计的调度算法的路由开销也会相应地降低，Overlay 层上的路由跳数将尽量接近 IP 层上的实际路由开销，算法的相

对平均延迟开销(Relative Delay Penalty, RDP)很低，调度效率也将大大提高。RDP 为 Overlay 网络逻辑跳数(Hop)与底层物理跳数(IP 跳数)之间的比值。

7.1.2 网格任务描述方法

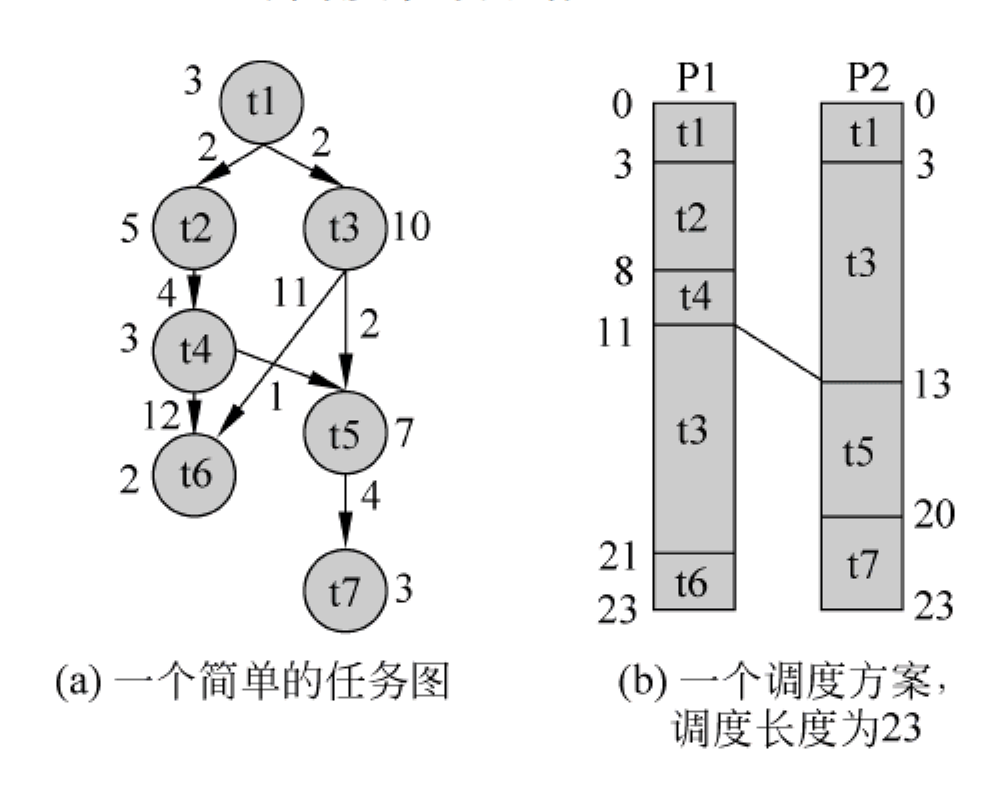
本节将对任务请求的描述方法进行研究讨论。在进行任务请求描述方法研究时，除了考虑如何精确表示任务数据集、用户的 QoS 要求等基本要素外，还应该考虑如何简单、明确地发掘与网格资源信息之间的关系，以提高资源调度算法的效率。在研究了描述并行任务的 DAG 模型的基础上，本节提出了使用有向无环图进行网格任务描述的方法，该方法可以简单地描述任务数据集和用户的 QoS，并且和第 6 章提出的网格资源信息有向图表示方法有很高的相似性，这将利于后续的基于匹配的网格资源调度算法的研究。

1. 基本 DAG 模型简介

通常并行程序使用一个加权的有向无环图(DAG)表示，图中的结点表示任务，边表示任务间通信以及数据依赖关系。形式化表示为：$G=(V,E,C,M)$，结点 $v\in V$ 表示一个计算开销为 $C(v)$ 的任务，有向边 $(u,v)\in E$ 表示任务的先后关系，即任务 u 完成后，任务 v 才能开始，另外，任务 u 完成后会发送消息给任务 v，其通信开销为 $M(u,v)$。如果这两个任务分配到同一个处理结点上，则认为通信开销为 0。如果存在一条从 u 到 v 的路径，则称 u 为 v 的前驱结点，v 为 u 的后继结点。特别的，对于 $(u,v)\in E$，称 u 为 v 的立即前驱结点(计为 $u\in \mathrm{IPred}(v)$)，v 为 u 的立即后继结点(计为 $v\in \mathrm{ISucc}(u)$)。没有任何前驱结点的结点称为入结点(Entry Node)，而没有任何后继结点的结点称为出结点(Exit Node)。并行程序的通信计算比(Communication-to-Computation-Ration, CCR)定义为任务图中平均通信开销比上平均计算开销[1]。

结点 v 的高度记为 height(v)。height(v)计算如下：如果该结点没有前驱结点，其高度为 0；否则，其高度为所有前驱结点高度值的最大值加 1。

例如，在图 7-1(a)中，任务的权值和边的权值分别表示为任务的计算开销和任务之间的通信开销。任务 t1 的计算开销为 3，而从 t1 到 t3 的通信开销为 2。t1 是 t3 的立即前驱结点，t3 是 t1 的立即后继结点。t1 是入结点，而 t6 和 t7 是出结点。其通信计算比 CCR 为 1.01。任务 t1、t2、t3、t4、t5、t6、t7 的高度值分别是 0、1、1、2、3、3、4。



(a) 一个简单的任务图　(b) 一个调度方案，调度长度为23

图 7-1　调度示例

图 G 的调度记为 $S(G)$，是将任务映射到处理结点上，并分配每个任务的开始时间，例如任务 $v \in V$ 上在处理结点 $p(v) \in P$ 的开始时间为 $\mathrm{ST}(v,p)$。这样，任务 v 在处理结点 p 的完成时间 $\mathrm{FT}(v,p)$，可以从公式 $\mathrm{FT}(v,p)=\mathrm{ST}(v,p)+\mathrm{C}(v)$ 计算得到(如果没有将 v 分配到 p，那么执行时间 $\mathrm{FT}(v,p)=0$)。一个结点可以映射到几个处理结点上，即任务复制。调度 s 的长度，定义为所有任务完成时间的最大值，即 $\mathrm{makespan}(S)=\max\{\mathrm{FT}(v,p) \mid v \in V, p \in P\}$。

例如，图 7-1(b)所示就是图 7-1(a)中 DAG(有向无环图)的一个调度，其中，任务 t1 和 t3 都被复制，调度长度 makespan 为 23。调度目标是，对于给定的 DAG 图 $G=(V,E,C,M)$，寻找具有最小调度长度的一个调度方案，即 $\mathrm{Min}\{\mathrm{makespan}(S)\ \forall S\}$。

在用 DAG 模型对并行程序建模过程中，程序中的循环不能显示出来，但是可以利用循环展开的方法将循环分解为多个任务[2]。对于多数数值计算程序，循环的上下限基本上在编译时都可以获得。而且，对于多数科学工程中的算法，其中的分支和不确定性很少。因此，DAG 模型可以比较精确地表示并行程序。

结点的权值(任务的计算开销)和边的权值(任务之间的通信开销)可以通过基准测试结合程序中的操作数等信息来估计[3,4]。

2. 网格任务的 DAG 表示

上述模型是基本的 DAG 模型，在做研究时可以根据具体的实验环境、不同的假设得到不同的 DAG 模型。本节研究的是网格计算环境中的任务表示模式，因此，根据网格并行任务的特点，设计了网格环境中任务的 DAG 表示形式。

参考文献[5]将分布式并行计算分为两种模式：一种模式是协作模式，将任务分解成一些独立的子任务，计算过程中，各个子任务需要其他子任务的中间结果，它们之间需要同步，确保子任务的同时开始和同时结束；另一种模式是非协作模式，分解后的各个子任务在相应的工作站上执行，中间不需要进行同步，不需要交换彼此的中间结果，最后将子任务执行结果汇聚成最终结果。大量研究表明，适合在网格中运行的任务应该具有高并行度、各个子任务之间通信量低的特点，因为网格资源的地理分散性和异构性使得网络通信开销成为影响网格性能的一个重要因素。当前大部分成功的网格应用项目往往用来处理非协作模式的并行计算，例如，SETI@home[6,7]。因此，本节主要进行非协作模式的网格并行计算研究。

首先将用户提交的网格任务划分成多个并行的子任务，划分方法可以根据具体的网格任务特征选择使用不同的算法，例如并行高斯消除算法[8]等。然后使用有向无环图(DAG)来表示这些并行子任务，这是一个 DAG 模型的拓扑排序过程。因为子任务之间相互不耦合、不通信，并行度高，因此此 DAG 模型可以转化成一棵 n 叉树[9]。

(1) 树的特征。此树是一棵高度为 2 的 n 叉加权树。在网格系统中，进行任务并行化的结点往往会负责收集此任务的各个子任务的计算结果，进行后续的处理。在研究具有超级结点对等网络的网格时，可以确定 n 叉树的高度为 2。

(2) 根结点的含义。根结点对应进行任务并行化和收集结果的超级结点，是网格任务的开始。根结点的权值代表进行任务并行化和结果收集所需的单位计算资源的数量。

(3) 叶子结点的含义。叶子结点对应不同的子任务，其权值代表完成此子任务需要的单位计算资源的数量。

7.1.3 基于树匹配的网格资源调度算法研究

在对网格资源信息表示方法、任务的描述形式进行了相应分析的基础上，本节提出基于树匹配的网格资源调度算法 nTreeMatch。该算法充分利用了网格资源信息表述方法和网格任务描述方式二者之间的相似性，利用匹配的方法来实现网格的资源调度。

1. nTreeMatch 算法的设计思想

网格资源信息用一个加权的有向图模型表示，即 $G=(V,E)$；图中的每个结点 V 一一对应网格结点，包括超级结点和客户结点；V 的权值代表某个网格结点当前可提供的计算资源的数量。

网格并行任务用一棵 n 叉加权树表示，树的根结点对应进行任务并行化和收集结果的网格结点，叶子结点对应不同的子任务；根结点的权值表示进行任务并行化和结果收集需要计算资源，叶子结点的权值表示完成此子任务需要的单位计算资源的数量；在子任务之间相互不耦合、不通信的情况下，n 叉加权树的高度为 2。

因为网格资源信息和并行任务最终使用加权有向图和加权树来描述，所以可以将网格资源的调度问题转换为两种数据结构间的匹配问题，即通过图和树之间的匹配算法进行网格资源的调度。

在这两种加权的数据结构的匹配过程中，可以借助有向图的最小生成树来简化匹配算法。因此，首先为加权有向图表示的网格资源 Overlay Network 找到一棵最小生成树，此最小生成树也是一棵加权树，树上结点的权值代表网格结点当前可以提供的计算资源的数量，边表示结点间的连接情况。

至此，网格资源的调度问题可以通过两棵树的匹配来完成，即网格资源 Overlay Network 的最小生成树和树高为 2 的网格任务 n 叉树之间的匹配。

设计资源调度算法的过程中，需要考虑以下的两个问题：

(1) 正确选择维护网格资源信息 Overlay Network 拓扑的协议。

(2) 正确选择路由转发协议。

好的路由选择机制能够有效控制路由信息的数量，降低网络路由和存储空间开销，提高算法的效率。好的拓扑协议可以对网络中的资源信息和路由信息进行高效的管理，另外，如前所述，如果在生成 Overly Network 拓扑时考虑网格结点的物理连接，生成含有最短路由信息的路由表，那么在进行资源调度时，路由表能够反映资源结点物理网的实际连接情况，路由开销将尽量接近物理网的实际路由，从而进一步提高资源调度算法的效率。

1) 网格资源 Overlay Network 拓扑协议

现阶段，有结构化和非结构化两种对等网络的拓扑协议可以用来维护这个资源 Overlay。此 Overlay 采用超级结点对等网络的结构，超级结点数量相对较少，并且相对稳定，因此，两种方式都是可以考虑的协议。基于对两种拓扑协议的分析，必须综合考虑拓扑协议对应的路由协议对资源调度产生的影响。

非结构化路由协议往往采用 Flooding 方式进行结点间的消息通信，在进行资源调度时，这种方式必将带来过多的网络流量，网络带宽被大量的广播消息所占据，导致系统整体性能急剧下降。此外采用 TTL 值来限制消息传播的半径，可能导致目标结点在传播半径以外时的情况下不能得到此消息，系统的可扩展性不好。

在进行资源调度相关的消息传播时,使用分布式哈希表 DHT 不会产生过多的网络流量,因此通信开销相对较低,这一点对于提高网格系统的性能尤为重要。同时,根据定义 6-2 和定义 6-4,使用有向图表示的网格资源信息 Overlay 层上的每个结点都包含一个路由表和信息表,记录本地资源的特征和位置信息。超级结点仅维护其客户结点的本地资源信息,并不对其他超级结点的资源信息进行备份,信息更新时客户结点仅需要通知其超级结点,而超级结点不需要向其他邻居超级结点转发该消息。由此分析可得,并不需要使用 DHT 进行结点信息表维护,也就是说,在 DHT 协议维护的 Overlay 中并不会存在大量的资源信息缓存备份。所以,可以仅考虑使用 DHT 来维护覆盖网超级结点的路由表。

基于以上分析,我们采用结构化的协议来维护此资源 Overlay Network 的拓扑结构。但是,与传统的分布式哈希表 DHT 协议不同,例如 Chord、CAN 等,本节仅使用 DHT 协议来维护网格结点的路由表,而不对结点的信息表进行 DHT 维护。即,当有超级结点加入或退出网络时,使用 DHT 协议调整结点的路由表;当网格结点进行资源信息更新时,例如,当结点的工作负载发生了变化时,仅需要采用 push 方式修改相应的客户结点和超级结点的权值即可。

2) 路由转发选择

在确定使用结构化协议来维护网格资源 Overlay 网络拓扑之后,接下来考虑选择何种路由转发机制来进行资源调度时相关信息的传播。

路由算法是 P2P 系统的核心,算法的优劣直接关系到 P2P 系统的核心性能如可扩展性、可靠性、可用性等。在设计路由算法时,需要考虑以下几个问题:

(1) 路由状态效率折中。状态指路由表所需要维护的邻居状态数目,效率指路由路径长度所代表的路由效率。由于状态效率对于 P2P 系统的性能具有本质性影响,因此,状态效率折中的研究一直是 P2P 系统路由的热点问题。一个重要的目标就是研究是否能够实现“$O(1)$的邻居数和 $O(\log_2 n)$的路径长”的状态效率,即“常量度”路由问题[10]。

(2) 容错性。由于结点在 P2P 系统内自由地加入和退出,因而在设计路由算法时容错性是一个重要的考虑因素。系统要在路由结点失效或者退出时,仍然能够保证路由可行性和正确性。可以使用 Static Resilience 来评估路由算法的容错性能。Static Resilience 指当结点失败并没有时间让别的结点重建别的邻居作为补偿时,即邻居结点知道一个结点失败了但并不与别的结点建立任何新的邻居关系时,路由的可行性及效率。参考文献[11]中认为 Static Resilience 与路由 Geometry 密切相关,环结构对于 Static Resilience 有效,此外还将路由表的邻居分为规则和持续两类,其中持续邻居对于 Static Resilience 更有效。

(3) 路由热点。路由热点指当存有资源的结点收到太多的任务请求时,此结点的出入路由流量太大,引起所谓的“热点”问题,即路由拥塞,结点会由于负载过重而反应不过来。路由热点的解决方案一方面可以通过有效的复制和缓存策略[12,13],以分流对于热点资源的请求,另一方面是采用负载均衡技术。在 DHT 中,负载均衡可采用虚拟服务器(Virtual Server)的方法,根据结点的能力分配负载,如 Chord。参考文献[14]中基于 Chord 进一步考虑了虚拟服务器根据动态负载的情况进行迁移的技术,重载结点可以将一些虚拟服务器迁移到轻载结点,同时考虑了系统的总体性能均衡。

(4) 物理网络匹配。物理网络匹配指 Overlay 网络应尽可能与物理网络相匹配以减少结点通信的开销和路由延迟,即使得 RDP 尽量小。这个问题的研究可以大致分为两种不同

的方法：一种是利用 Internet 的层次信息，如自治系统 AS、IP 前缀进行拓扑适应构造，这是一种直接与物理网络匹配的方法；另一种是间接的匹配方法，即采用路由选择技术。针对第一种直接匹配的方法，eCAN[15]借助 BGP 表来引导路由，使得结点通信本地化，减少通信开销和路由延迟；参考文献[16]中利用 IP 前缀使得路由充分利用 Internet 层次信息，并模拟路由层的最短路径算法以匹配物理网络。这些方式都直接基于 Overlay 网络的拓扑结构。

采用路由选择技术的间接匹配方法的研究主要集中在 DHT 协议领域，目前可以分成三种不同的技术：邻近邻居选择（Proximity Neighbor Selection，PNS）、邻近路由选择（Proximity Routing Selection，PRS）和邻近标识选择（Proximity Identity Selection，PIR）。PNS 在构造路由表时，结点选择与该结点邻近（比如时延短、物理位置靠近）的结点作为邻居。这种优化策略选择结点的物理邻近结点进行路由跳转，从而使得路由间接与物理网络匹配。该方法有好的延迟伸展、负载平衡和本地路由收敛特性[17]，但是该方法的限制是不支持如 CAN 和 Chord 这类要求在路由表中明确指出标识空间的下一个结点标识的 DHT 算法的。PRS 在路由时选择与该结点邻近的邻居作为下一跳。与 PNS 相反，PRS 在路由过程中进行动态选择。假设每一跳都有 k 个结点可供选择，则每一跳的平均延迟可以从原来的网络中任意两个结点延迟的平均值减少到网络中任意一个结点到其他任意 k 个结点延迟最小值的平均值，所获得的性能提高同 k 的大小成正比，增大 k 的值意味着每个结点路由表的增大，从而需要更多的资源消耗以维持链接。除此之外，一味考虑选择延迟最低的结点转发查询也可能导致路由逻辑跳的数目变大。

PIR 的工作原理是：如果标识与物理位置相关，那么在以标识为基础的 Overlay 网的路由就能够尽可能接近以 IP 为基础的物理网的路由，因此在新结点加入时，产生的结点标识与结点所在的物理位置相关。常用的 PIR 方法有 Landmark[18]技术：取 m 路标，然后每个加入的结点通过 ping 这 m 路标得到的值排列成 m 位，即为结点标识。由于此标识反映了与 m 路标的物理位置关系，因此，将结点标识与物理位置结合了起来。参考文献[19]实现了 PIR 路由，它在每一跳获得了 IP 延迟的两倍或者更少的性能。然而，该方法也存在一些缺点：首先，它破坏了标识空间的均匀分布，在 Overlay 网中容易造成负载不均衡问题；其次，由于映射算法的限制，该方法在一维空间中，如 Chord、Tapestry 和 Pastry 等，工作效果较差；再次，标识空间中相邻的结点由于物理也相邻，容易造成并发失败，而由于在 Chord 和 Pastry 这样的系统中结点在邻居中复制数据，还存在安全性和鲁棒性问题。

（5）异构性。当前的 P2P 路由算法在处理 Overlay 层的消息路由时并不考虑结点的处理能力差异，例如网络带宽、存储空间和可用 CPU 数目等。然而，研究表明由于部分结点的性能瓶颈可能导致路由算法失效[20]。因此，在设计路由算法时需要考虑结点的异构性问题。利用异构性将更多的任务分配给有高网络带宽、大存储容量和好的 CPU 处理能力的结点。参考文献[21]建立了一个超级结点虚拟层，并将本地结点组织成一个以超级结点为中心的组，采用两阶段路由，第一阶段是将请求路由给超级结点层，第二阶段时再将请求路由给超级结点所在组的目标结点。这种方式可以避开低能力结点的瓶颈，提高通信效率，降低通信延迟。

在设计 nTreeMatch 算法的路由转发机制时考虑了上述因素，DHT 协议在维护结点的路由表时有很明显的优势，例如，在 Chord 中，每个结点需要保存约 $\log_2 N$ 个其他结点的路

由信息(N 为网络中的结点数),在 CAN 中,每个结点只需维护 $2d$ 个邻居结点的信息,当有结点加入或退出网络时,两种协议需要调整的路由表工作量都很少。因此,在选择路由协议、设计结点路由表时,我们选择使用简单、正确、性能好、易维护的 Chord 协议作为路由协议的设计基础。但是,Chord 协议存在一些问题,例如,其 Overlay 层上的路由没有考虑物理网实现(IP 层路由),因此,有可能存在 Overlay 层上的路由逻辑跳数不多,但是实际的物理跳数(IP 跳数)却很多,从而导致资源调度时间过长,使得调度算法的性能不能达到最优。所以,根据网格资源 Overlay 结点的特性,对 Chord 协议进行了改进,使得算法的相对平均延迟开销 RDP 尽量小。

在进行资源调度,即两种树形数据结构进行匹配时,当某个子任务结点无法在资源 Overlay 网络拓扑的最小生成树的某结点上获得满足计算要求的资源信息时,此子任务的请求将通过一个路由函数转到另一个资源结点上开始新的匹配。在转发此任务请求时,路由函数通过查找当前结点的路由表,找到其最近的邻居结点,然后将任务请求转发过去。

为了弥补 Chord、CAN 协议的不足,Pastry、Tapestry 等另外的几种结构化协议在算法设计时考虑了查找的 IP 路由和逻辑跳之间的关系,通过一定方式使得 RDP 尽可能少。算法中,每个结点的路由表中的结点到该结点的物理距离都尽可能短,从而保证查找过程中实际物理跳数也尽可能的少。算法的最大优势在于:物理距离与逻辑距离成正比。但是,Pastry 和 Tapestry 的路由表无法达到全局一致性,算法实现相对复杂。Tapestry 中,更新阈值的选择是个挑战,选择值过大可能导致整个系统的路由性能逐步劣化,选择值过小,更新所需的开销将难以承受。Pastry 中,三角不等式原理本身不太符合网络的实际情况,以此为基础的路由优化值得怀疑。当结点加入或退出系统时,两者的系统开销都较大。

基于网格 Overlay 拓扑结构的设计,我们认为,如果在构建 Overlay 拓扑的最小生成树时正确考虑 IP 层路由跳数,算法的路由耗费、RDP 应该能够降至可以提高网格资源调度性能的范围。因此,在构造 Overlay 拓扑时利用了网格结点的 IP 前缀,这是一种 Overlay 网络直接与物理网络匹配的方法,基于如果每步路由都是最短时延,那么总体时延也应该得到优化的原理。我们利用网络临近原则来构建这棵最小生成树,使用参考文献[22]中的机制来发现拓扑临近的邻居结点。这样,存储在结点路由表中的邻居结点是物理路由跳最小的结点,算法的 RDP 会非常小,Overlay 层上的路由耗费可以最大限度地接近物理网络上的真实路由。具体实现方法如下。

在构造 Overlay 拓扑时,如果一个新的结点加入到此网格,此结点需要选择一个和它是网络临近的结点作为其邻居结点。提取 BGP 路由表中的地址前缀信息,如果两个结点的 IP 地址共享一个共同的地址前缀,那么我们认为这两个结点是网络临近的。一个结点使用这种方式来寻找其网络临近的邻居结点。同时,我们认为,如果一味考虑选择延迟最低的结点来转发任务请求查询可能导致路由逻辑跳的数目变大。因此,如果一个结点不能够从路由表中找到 IP 地址前缀能和其相匹配的结点,那么此结点将从路由表中随机选择一个结点作为其邻居结点。

网格资源调度算法的路由表结构以及超级结点的加入和退出时的处理方法与 Chord 相同。表 7-1 所示给出了 nTreeMatch 的路由表结构。表中具体的参数含义可参考 Chord 算法描述[23]。

表 7-1 nTreeMatch 路由表

字　段	结构及意义
finger[k]. Start	$(n+2^{k-1})\bmod 2^m$，$1\leqslant k\leqslant m$
. interval	[finger[k]. start，finger[$k+1$]. start]
. node	第一个 ID 大于 n. finger[k]. start 的结点
. successor	ID 空间上的后继结点
. predecessor	ID 空间上的前驱结点
. redirect	重定向结点 $1\leqslant k\leqslant m$

2. 算法定义

基于广度优先查找(Breadth First Search,BFS)算法,下面给出通过两个树形数据结构之间的匹配进行网格资源调度的算法 nTreeMatch。为了方便算法描述,在阐述 nTreeMatch 算法的工作原理之前,首先给出以下几个定义。

定义 7-1 资源树。Overlay 网络拓扑的最小生成树称为资源树,用 R 表示。资源树 R 反映了系统的实时工作状态,例如网络连接和实时工作负载状态。$|R|=m$ 表示树上的结点个数为 m,m 为自然数。

定义 7-2 任务树。网格中的并行任务树称为任务树,用 T 表示。资源树 T 反映了将任务并行化为多个子任务的情况,根结点的权值表示进行并行化、结果收集以及处理各个子任务需要的计算资源数量。$|T|=n$ 表示树上的结点个数为 n(n 为自然数)。

定义 7-3 树的初始化。进行两个树形数据结构的匹配之前,首先进行树形数据结构的初始化操作,具体内容是将每一棵树上高度相同的叶子结点根据权值大小按照升序排序。进行初始化时使用 BFS 算法遍历每棵树上的所有叶子结点,然后将处在同一层中的叶子结点进行升序排序即可。

定义 7-4 树形数据结构的方法 NodelistBFS(s)。假设 Ω 是一棵已经初始化的树的结点集合,例如,树 R 的结点集合为 $R=\{R_1,R_2,R_3,\cdots,R_m\}$,树 T 的结点集合为 $T=\{T_1,T_2,T_3,\cdots,T_n\}$。NodelistBFS($s$)方法可以获取结点 s 的儿子结点所在层的所有结点排序列表。此方法的实现过程中使用了 BFS 算法。

定义 7-5 结果集合。两棵树的匹配结果为一组或多组可以匹配的结点对,使用结果集合 MNP 来存储这些可以匹配的结点对的列表,例如,$\text{MNP}=\{\{T_1\to R_1,T_2\to R_3,\cdots,T_n\to R_m\},\cdots,\{T_1\to R_3,T_2\to R_4,\cdots,T_n\to R_m\}\}$代表任务树结点 T_1 和资源树结点 R_1、任务树结点 T_2 和资源树结点 R_3 以及任务树结点 T_n 和资源树结点 R_m 是一组匹配的结点集合,而任务树结点 T_1 和资源树结点 R_3、任务树结点 T_2 和资源树结点 R_4 以及任务树结点 T_n 和资源树结点 R_m 是另外一组可以匹配的结点集合。

nTreeMatch 算法的伪代码如下:

```
nTreeMatch (Tree &R, Tree &T) {
    X = FindSubtreeRoot(Tree &R, Tree &T);
    R' = NodelistBFS(X);
    T' = NodelistBFS(root of T);

    Do {
```

```
        If (Compare (R'(j).weight, T'(i).weight) {      //True for >= , False for <;
            T'(i) and R'(j)  -> MNP;
            T'(i) = i.RightBrother;
        }
        Else {
            R'(j) = FindSuccessor (R'(j));  // find the successor of j in Chord route table;
            Compare (R'(j).weight, T'(i).weight);
        }
    } Until (T'(i).RightBrother≠NULL);

  For each node in R' and T'{
      If Min(Σ(unmatched node in T').weight) then{
            all the matched nodes pairs  -> MNP;
            For each element in MNP {
                If Max (Σ(unmatched node in R').weight) then
                    this element be the only one in MNP;
    }}}
    For each node in T'{
        if (all the nodes in T').matched = true then
            return MNP;
        else{
            R'' = R'.delete (unmatched nodes and their children);
            T'' = T'.delete (matched nodes);
            nTreeMatch (Tree &R'', Tree &T'');
}}}

FindSubtreeRoot (Tree &R, Tree &T){
    For each node in BFS(Tree &R){
        If weight (x) >=  weight (root (T)) then
            Return x and Break;
}}
```

在网格环境中，进行任务并行化的结点往往也会将返回的各个子任务的处理结果收集起来，也就是说，任务树的根结点代表的任务将由资源树上的一个结点处理。同时，任务的并行化是网格系统处理用户任务的第一步，所以调度过程首先从为任务树的根结点找到可以匹配的任务树的结点开始。这个步骤对应函数 FindSubtreeRoot (Tree &R, Tree &T)。nTreeMatch 调度算法使用该函数来寻找资源树上可以进行匹配的一棵资源子树，以此来保证任务树 T 的根结点可以首先从资源树中找到匹配的处理结点。

在进行资源调度，即两种树形数据结构进行匹配时，当某个子任务结点无法在资源子树上的某结点获得满足计算要求的资源信息时，该子任务请求将通过一个路由函数 FindSuccessor 转到另一个资源结点上开始新的匹配。在转发此任务请求时，该函数通过查找当前结点的路由表，找到其最近的邻居结点，然后将任务请求转发过去。该路由表生成时利用了上面提到的网络临近原则，通过提取 BGP 中结点 IP 前缀信息得到物理层上路由跳数最小的邻居结点列表。

函数 Min(Σ(unmatched node in T'). weight)可以判断结果集合 MNP 中一个匹配的结点对列表能否使得在资源树 T 中那些没有找到合适的匹配结点的结点的权值最小。这样做的目的是为了确保尽可能多的子任务可以分配到计算资源。在结果集合 MNP 中可能存

在一个或多个满足此条件的元素,但是为了保证资源调度的有效性,使得任务使用尽量少的资源、网格中的资源不被耗尽,只有使得资源树 R 中剩余的结点的权值最大的元素,即MNP 中的已匹配结点对列表可以被最终确定为资源调度的最优结果。函数 Max(Σ(unmatched node in R'). weight)可以达到上述目的。

R. delete (x)的代码设计基于定理 7-1。

定理 7-1 *在第一轮匹配过程中,如果任务树 T'上的一个结点 n 没有在资源树 R'上找到匹配的结点,那么在执行第二轮匹配时,此结点 n 的匹配范围限制在由已经资源树 R'上那些已经找到相关匹配的结点的孩子结点组成的集合。*

证明:此定理是显而易见的。如果任务树 T'上的一个结点 n 在第一轮匹配过程中没有找到可以匹配的资源树 R'上的结点,说明资源树 R'中这一层上所有剩下未匹配的结点都不能满足结点 n 的要求。假设结点 n 可以在 R'未匹配的结点的孩子结点中找到匹配结点 m,但是必须为结点 m 和 R 上其他结点的通信支付昂贵的开销耗费。因为此通信过程必须通过 R 的根结点进行,那么此路径将成为所有匹配中路径最长的一条,相应地,其路由耗费也将是最高的。因此,在二次匹配过程中,仅仅考虑已经找到相关匹配的结点的孩子结点,而不再考虑那些未匹配的结点的孩子结点。证明完毕。

需要说明的是,在将任务划分成多个并行子任务后,如果在一定时间内没有返回计算结果,调度策略认为此任务计算失败,然后将任务重新分配,如果有效时间内返回了多个计算结果,则取最早返回时间的结果为有效结果,将其他结果丢弃。这种做法可能会浪费一定的计算资源,但是却简单、巧妙地避免了使用结点间的进程迁移来提高调度策略容错性的复杂性。

7.1.4 算法时间复杂度分析

nTreeMatch 调度算法的核心是匹配过程,由两个主要的步骤组成:①树的遍历和排序;②路由跳转。下面将分别对匹配过程的两个主要步骤的时间复杂度进行分析,最终获得算法的整个匹配过程的时间复杂度。其中 n 为结点的数目,$n=\max(\text{NodeNumber}(T), \text{NodeNumber}(R))$。

1. 树的遍历

在找到和任务树的根结点匹配的资源树结点之后,算法开始。按照广度优先对树进行遍历和排序,遍历每步的计算步骤为

$$G_i(n)\begin{cases}\leqslant 1+2\times(i-1), & \text{子结点数目大于 1}\\ =1, & \text{其他}\end{cases}$$

极端情况下只需对样树上的所有结点进行一次扫描,即所用时间为 $O(n)$。因此,遍历的总的计算步骤:$n\leqslant G(n)\leqslant\sum_{i=1}^{n}(2i-1)$,即

$$n\leqslant G(n)\leqslant n^2$$

2. 路由跳转

考虑算法的路由过程,算法时间复杂度和 Chord 算法一样,$O(\log_2 n)$。所以,算法的时间复杂度为 $O(\log_2 n\times n^2)$。

经过以上分析，可以得知 nTreeMatch 调度算法的匹配过程时间复杂度为多项式时间。由于各个步骤遍历的结点数不同，所以没有给出总时间复杂度的计算结果。

7.1.5　算法实验及结果分析

为了验证 nTreeMatch 调度算法的性能，我们基于仿真实验对调度算法的整体性能、平均服务质量 QoS、路由效率、空间开销、结点负载状态进行测试，并分别和 Min-min、Max-min 算法、Chord 算法做了相关比较。

1. 实验环境

基于 GT-ITM[24] Internet 网络，仿真程序使用 GT-ITM 网络拓扑产生器建立 Transit Domain 和 Stub Domian，以此来模拟基于超级结点对等网络拓扑的网格环境。

在网格模拟环境中，GT-ITM 网络拓扑分为两层：Transit Domain 和 Stub Domain。其中，Transit Domain 对应网格中的虚拟组织，Stub Domian 对应网格中的由超级结点维护的客户结点集合。GT-ITM 网络的网格规模为 28800，平均虚拟组织规模为 400。逻辑结构中的所有结点，包括超级结点和客户结点，都通过 SHA-1 安全哈希函数得到，路由表的行数为 8，列数为 8。结点在网络中随机分布。实验给每个结点赋予一定的生命周期，以此来模拟结点的加入和退出。网络连接分为局域网(LAN)和广域网(WAN)，实验中 WAN 的带宽均匀分布在区间[0.5，1.0](单位为 Mb/s)上，LAN 的带宽均匀分布在区间[2.5，5](单位为 Mb/s)上。但是，在实验中我们假设通信链路的带宽足够大，不考虑通信延迟。同时，我们还假设一条通信链路一次只能发送或接收一个消息。因为 nTreeMatch 算法在调度计算任务时，不考虑链路竞争，并忽略由此产生的相应的路由信息。在真实的网格环境中，一条物理通路可能同时被几条逻辑通路所复用。虽然算法不考虑链路的竞争以及路由，可能使得算法生成的调度结果与实际运行之间存在一定的误差，但是因为本节考虑的是并行任务的广域调度，并且调度任务的各子任务中间的通信量非常小，所以，和整个并行任务的执行时间相比，此误差可以忽略不计。同时，仿真程序会周期性更新客户结点的可用资源信息、索引信息，同时维护超级结点的路由表和邻居信息。使用的更新数据由 NWS 测定实际应用系统得到，包括结点负载、CPU 空闲率等。设定结点的计算速率按均匀分布随机分布在 200、400、600、800 和 1000(单位为 Mflop/s)之间。这些跟踪数据使得模拟环境能够仿真实际网格环境的资源异构性特征。

表 7-2 列出了仿真实验中使用的 GT-ITM 参数。其中，T 为网络中 Transit Domain 的个数，即虚拟组织的个数；N_T 为每个 Transit Domain 的结点数，即超级结点个数；S 为每个 Transit Domain 结点平均连接的 Stub Domain 的个数，即每个虚拟组织中的超级结点数；N_S 为每个 Stub Domain 内的结点数，即超级结点维护的客户结点个数；D_S、D_{TS}、D_T、D_{TT} 分别为 Stub Domain 内的物理链路长、Stub Domain 与 Transit Domain 间的物理链路长、Transit Domain 内的物理链路长、Transit Domain 间的物理链路长。

表 7-2　仿真实验中 GT-ITM 参数的设定

参数	T	N_T	S	N_S	D_S	D_{TS}	D_T	D_{TT}
参数值	3	8	3	8	1	3	5	7

实验中,任务的表现形式为对客户结点可利用资源信息的请求。仿真程序假定任务总数为1000,任务请求的发起过程服从泊松分布,平均到达间隔时间为8s。任务的输入数据大小相同,输入文件使用实际应用系统的数据,其中,设定任务对资源的需求量按均匀分布随机产生,分布区间为[0.1,100]。同时我们设定,任务一旦被调度后,不再执行再调度。

2. 实验结果

1) 整体性能

采用任务的总执行时间makespan作为nTreeMatch算法整体性能的评价指标。为了分析算法的性能,比较了nTreeMatch算法和Min-min算法、Max-min算法的性能。在实验中,对于同一个任务,分别记录使用不同的调度算法,任务需要的总的执行时间;然后改变任务输入数据的大小,进行了几组测试。图7-2记录了任务输入大小分别为1MB、2MB、3MB时,在三种不同的调度算法下所需的执行时间,每种情况的实验数据是同样条件下重复200次的平均结果。

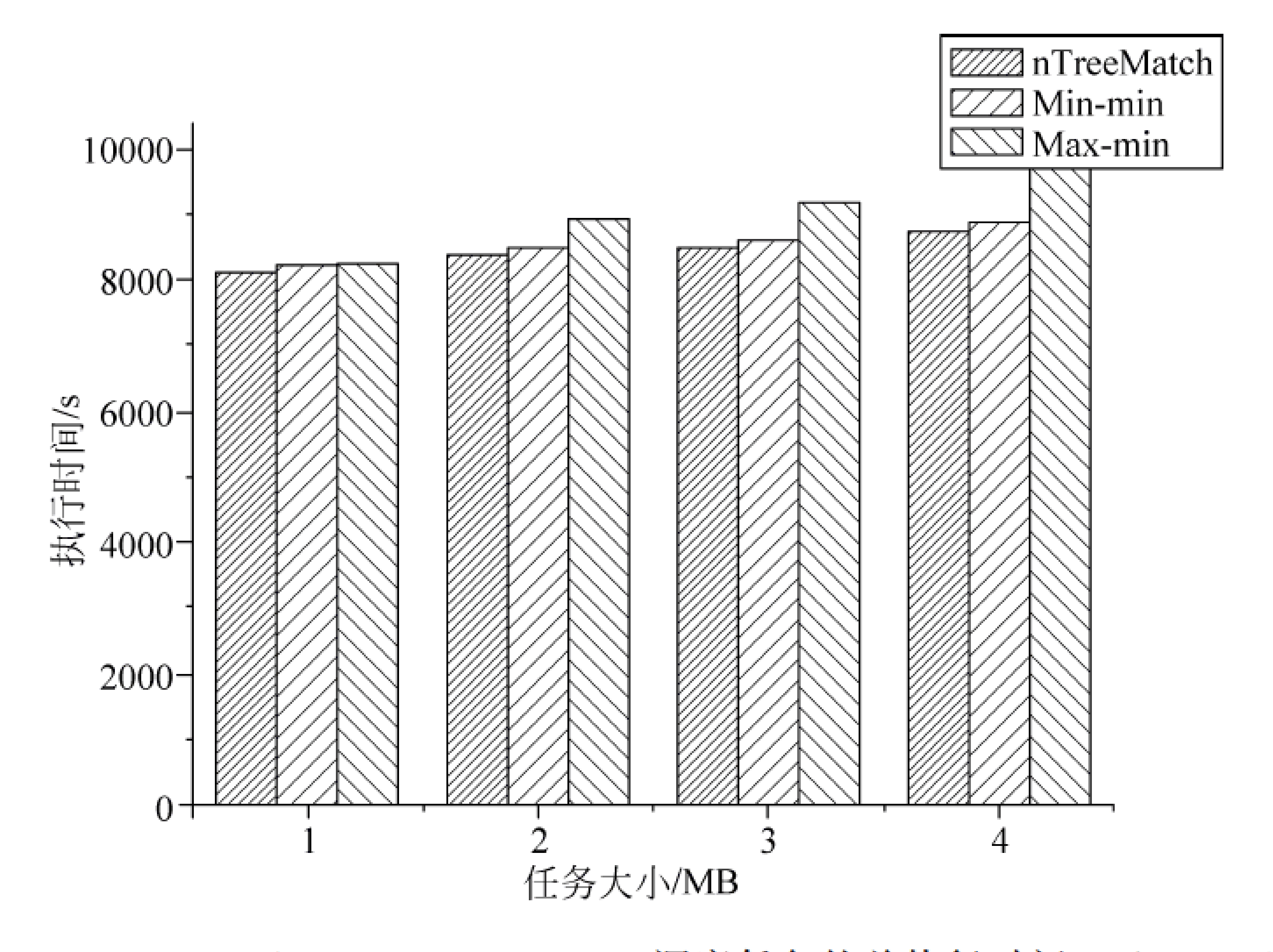


图7-2 nTreeMatch、Min-min、Max-min调度任务的总执行时间makespan比较

从图7-3可以看出,在同样的任务输入数据大小下,nTreeMatch调度算法使得任务的makespan最小。并且,输入数据越大,即任务的计算量越大的情况下,nTreeMatch调度算法越能缩短任务的makespan,呈现的性能越好。例如,1MB的任务下,使用nTreeMatch算法进行调度,任务的makespan为8137s,而Min-min、Max-min分别为8238s和8256s;而当任务输入增长为4MB时,nTreeMatch调度策略使得任务的makespan为8750s,而Min-min、Max-min分别为8953s和9768s。由此分析可以得出结论:通过各个网格计算结点间和相互协作,可以降低任务总的执行时间,说明了nTreeMatch调度算法的有效性。任务计算量越大,计算时间越短,说明了由于网格结点间通过Chord网络,建立了P2P的协作关系,加速了计算的进程。

2) 平均服务质量QoS

平均响应率是评价一个调度算法的服务质量QoS的参数之一,其他可以用来衡量服务

质量的要素还包括响应时间、吞吐率、可用性、安全性等。实验中采用平均响应率(Average Response Ration)作为评价 nTreeMatch 算法整体服务质量(QoS)的评价指标。对单个任务的 QoS,响应率(Response Ration, RR)的定义为

$$RR = 任务执行时间 /(任务完成时间 - 任务提交时间)$$

响应率反映了一个特定任务等待处理的时间期望,反映了算法对单个任务的服务质量。而平均响应率是一组任务的所有响应率的平均值,可以反映算法的整体服务质量。

在分析算法的 QoS 时,仍然将 nTreeMatch 算法和 Min-min 算法、Max-min 算法的 QoS 进行比较。在本实验中,对于同一个任务,分别记录在使用不同的调度算法情况下,任务的执行时间、提交时间和完成时间,计算任务的响应率;然后改变任务输入数据的大小,进行了几组测试。图 7-3 所示记录了任务输入大小分别为 1MB、2MB、3MB 时,在三种不同的调度算法下,重复执行任务 200 次后,获得的实验结果。

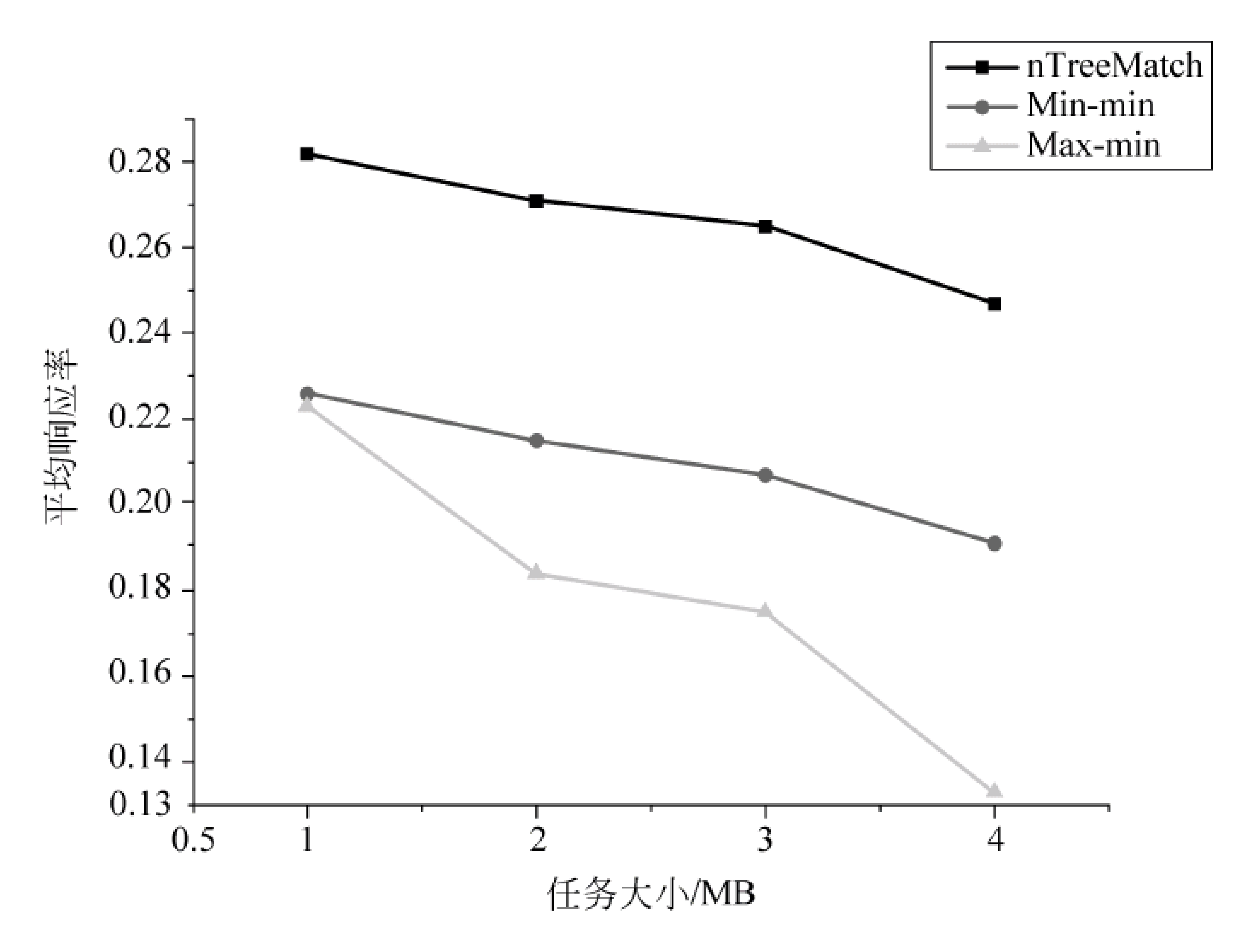


图 7-3　nTreeMatch、Min-min、Max-min 算法的平均响应率(QoS)比较

从图 7-3 可以看出,平均响应率随着任务输入数据的增加而变化。在同样的任务输入数据大小下,nTreeMatch 算法的平均响应率最大,明显高于 Min-min 算法和 Max-min 算法。例如,任务输入数据大小为 1MB 时,nTreeMatch 算法的平均响应率为 0.282,而 Min-min 算法和 Max-min 算法分别为 0.226 和 0.223。随着输入数据的增大,即任务计算量越来越大的情况下,三种调度算法的平均响应率均有所下降,其中 Max-min 算法的平均响应率下降最快,虽然 nTreeMatch 算法和 Min-min 算法的平均响应率下降趋势相似,但是 nTreeMatch 算法仍能保持较高的平均响应率。例如,当任务输入增长为 4MB 时,Max-min 算法的平均响应率很快下降到 0.113,Min-min 算法和 nTreeMatch 算法分别为 0.119 和 0.247。对于这种实验结果,可以给出以下解释:Min-min 算法和 Max-min 算法在进行调度时都仅考虑任务在某个结点的最小完成时间(Minimum Complete Time, MCT),而 nTreeMatch 算法主要考虑调度某个任务之后,还为其他任务预留更多的计算资源,在其他任务到达时可以得到及时调度,每个任务等待执行时间的开销小,有助于降低任务总执行时间,可以保证在任务高到达率时,算法仍能获得较高的响应率。由此分析可以得出结论:

nTreeMatch 算法的平均响应率优于其他两种算法。

3）路由效率

在实验中，记录 nTreeMatch 算法在转发任务请求时所需的路由跳数，然后和 Chord 算法在转发结点查询时的路由跳数进行比较，如果 nTreeMatch 算法的路由跳数小于 Chord 算法，则认为 nTreeMatch 算法的路由效率高于 Chord 算法。通过这种方法来测试 nTreeMatch 算法采用网络临近原则选择临近邻居结点来降低 RDP 的设计是否有效。测得的实验结果如图 7-4 所示。

图 7-4 所示为路由跳数的概率密度函数（Probability Density Function，PDF）。Chord 算法 PDF 最大时的路由跳数为 7，对应的 PDF 的值为 0.21；nTreeMatch 算法 PDF 最大时的路由跳数为 6，对应 PDF 值为 0.19。从以上实验数据注意到，nTreeMatch 算法的路由跳数虽然非常接近 Chord 算法，但是仍然略小于 Chord 的路由跳数。说明 nTreeMatch 算法采用的网络临近原则确实可以降低 RDP，虽然这种改变是轻量级的。分析其原因，主要是由于 nTreeMatch 算法转发任务请求时，首先会路由到所在域的超级结点，从而导致了路由跳数的增加，一定程度上影响了算法的路由效率。

4）空间开销

图 7-5 所示为在平均虚拟组织为 400 的网格拓扑上，客户结点数对空间开销的影响。分析实验结果可得，nTreeMatch 算法的空间开销略高于 Chord 算法，同时 nTreeMatch 算法的空间开销随客户结点数量的增加而增加。这是因为在 Chord 网络中，每个结点仅维护一个路由表，而网格中的超级结点还要维护其所辖域内的所有客户结点的信息。实验测得，当虚拟组织大小为 400 时、客户结点数为 80000 时，Chord 算法和 nTreeMatch 算法的空间复杂度分别为 0.39KB，0.41KB。nTreeMatch 算法的空间开销是完全可以接受的。

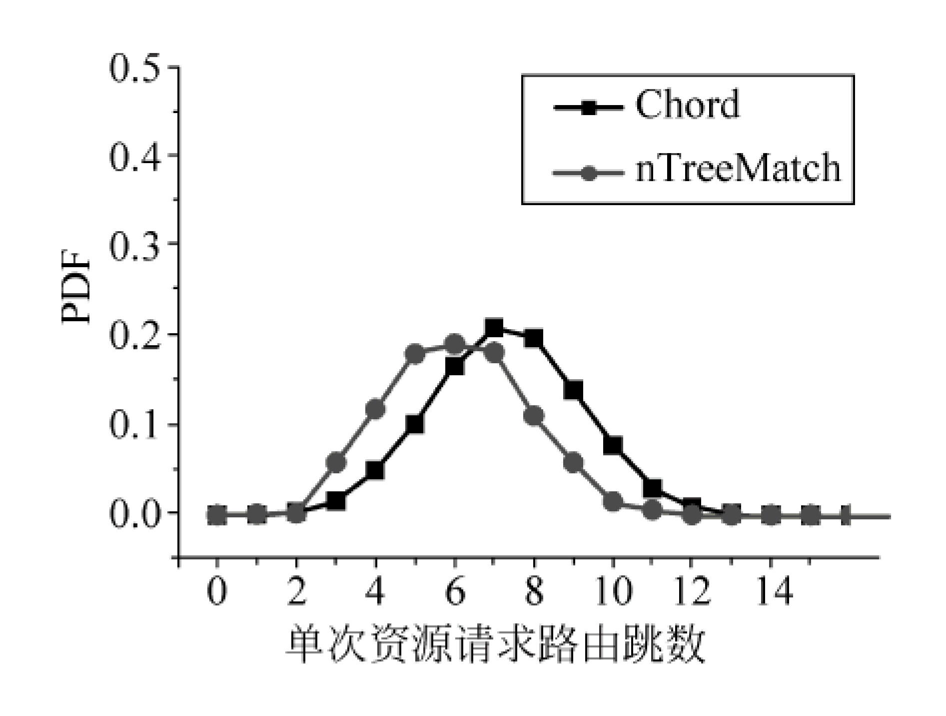


图 7-4　nTreeMatch 和 Chord 的单次资源请求路由跳数的概率密度函数

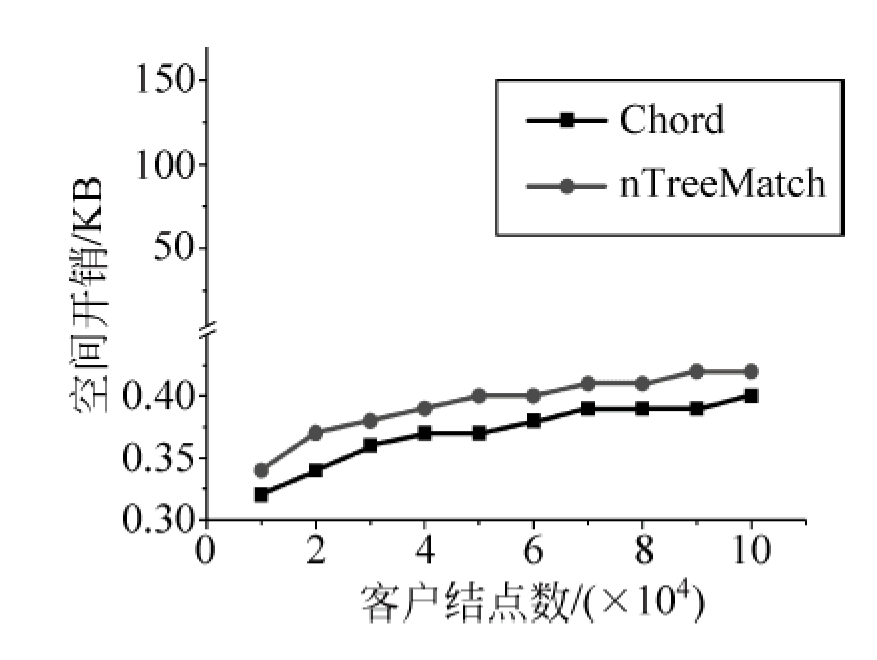


图 7-5　网格环境中的客户结点数对空间开销的影响

5）结点负载状态

是否能够根据网格资源结点的资源属性的动态变化，进行动态的网格任务调度是评价网格资源调度算法的一个重要测度，同时，结点负载状态及负载平衡也是评价对等网络资源管理方式的一个重要测度。为了评价 nTreeMatch 算法的动态调度性能，本部分的实验采用记录每个结点路由的消息数的方法来评价结点的负载状态。图 7-6 所示是 nTreeMatch 算法的结点负载的概率密度函数，在图 7-7 中，将 nTreeMatch 算法和 Chord 算法的结点负载的概率密度函数进行了比较。

为了能清楚地显示 nTreeMatch 算法的结点负载分布，图 7-6 所示中将纵坐标轴进行了分割，从图中可以看出有少量结点的负载非常高，概率约为 $3.5\times10^{-5}\sim7\times10^{-5}$。这些高负载的结点是网格中的超级结点。图 7-7 所示的实验结果表明 nTreeMatch 算法的结点负载比 Chord 算法的结点负载稍低。

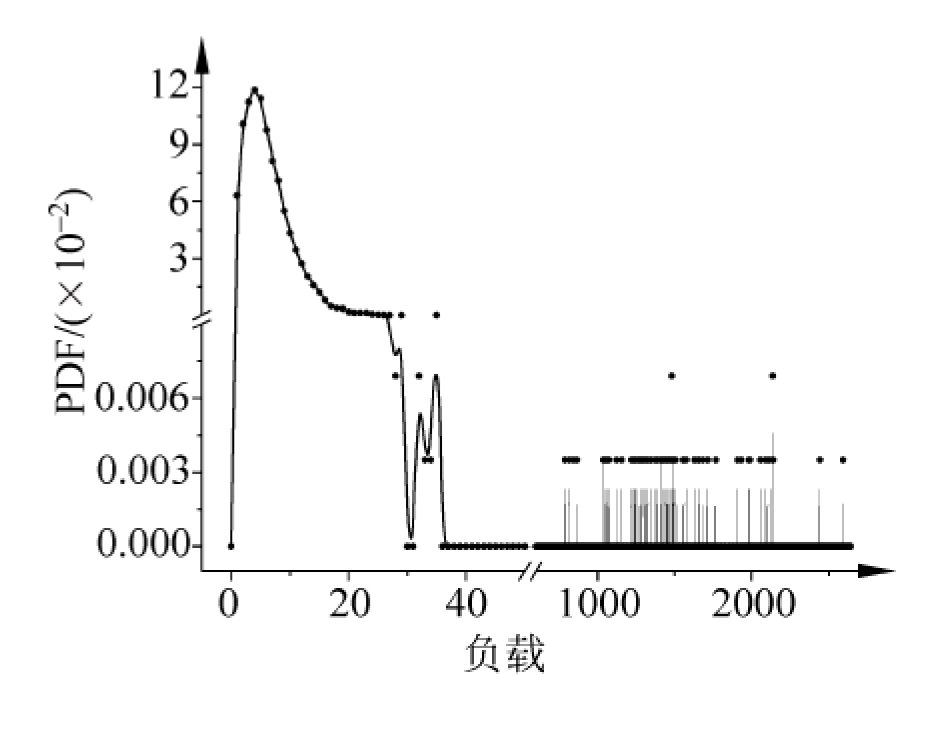


图 7-6　nTreeMatch 算法的结点负载的概率密度函数

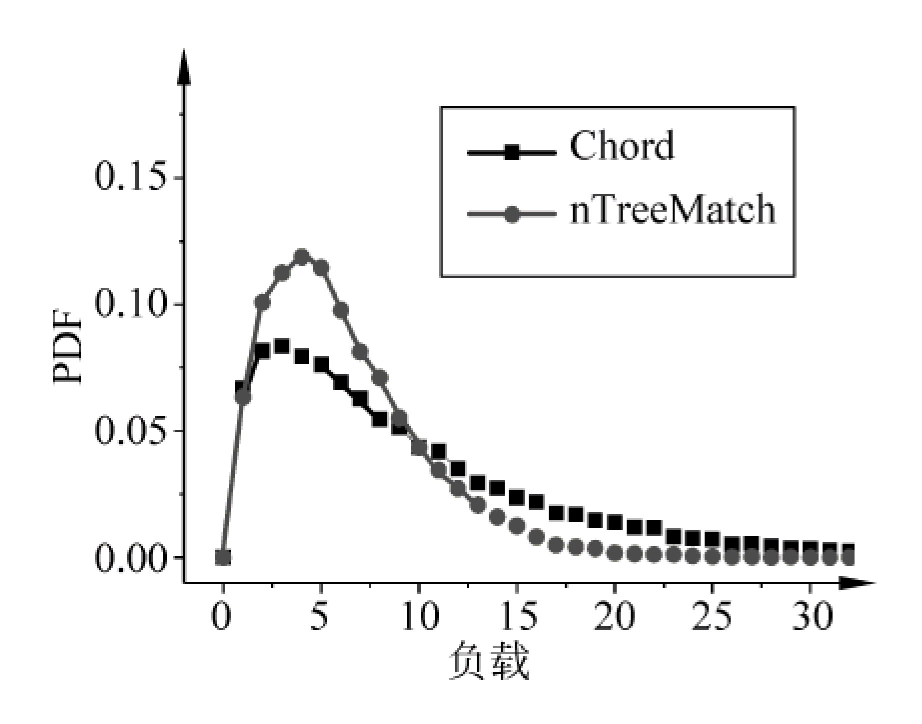


图 7-7　nTreeMatch 算法和 Chord 算法的结点负载概率密度函数比较

以上针对为解决任务并行度高的计算密集型科学应用而设计的专用计算网格提出了基于树匹配的网格资源调度算法 nTreeMatch。算法充分利用了网格资源信息有向图表示方法和网格任务 DAG 表示方法之间的相似性，将网格 Overlay Network 拓扑的最小生成树和 n 叉任务树进行匹配，以实现网格资源的实时调度。提供者的实时资源属性，进行动态的网格任务调度，可以获得较好的负载均衡效果。在构造 Overlay 时借助了网络临近原则同时算法充分利用了 Overlay 网络拓扑提供的路由信息，使用网络邻近原则使得 Chord 环上的邻居结点同时为物理网络上邻近的结点，从而以轻量附加开销来有效减少 Overlay 层上的路由跳数，使得 Overlay 层上的路由跳数尽量接近 IP 层上的路由跳数，降低 RDP，并最终达到提高算法效率的目的。大量的对比实验结果表明，算法可以有效地对网格并行任务进行动态调度，提供较高的服务质量。

7.2　基于资源发现的网格资源调度算法

在进行网格资源调度时如何获取动态变化的资源属性的最新状态，而不是过时的信息，是进行网格资源调度算法必须考虑的重要问题。本节针对现有的资源调度算法往往无法获取资源的实时信息等问题，提出了一个适用于基于 Web Service 的通用网格的资源调度算法，该算法将网格资源调度问题转化为分布式资源发现问题，在进行资源调度时可以获取资源的实时信息，实现网格资源的动态调度。

7.2.1　基于 P2P 的网格资源调度模型

网格计算的动态性表现在很多方面，静态属性方面的不同，例如操作系统版本、存储空间等；高度动态变化的因素包括可利用 CPU、存储空间、网络带宽以及其他工作负载信息等。网格调度算法必须考虑这些动态可变的因素，从而可以对资源进行更加有效的动态调

度。现有的资源调度策略，例如 AppLes、Condor-G、Nimord-G 等均采用信息收集的方式，即通过 LDAP 协议查询 Globus 或 OGSI 提供的底层 MDS 获得资源信息，然后进行调度。而网格资源信息的更新引擎依赖于服务器，其更新速度远落后于资源信息的变化，容易出现信息过时的现象。信息的实时性是评价网格调度策略的一个重要的评判标准。与集中式的资源管理相比，对等网络的最大优势在于分布的资源发布和发现技术。发现过程不受宿主设备的限制，可以对网格进行高效、全方位的搜索。搜索深度和广度在理论上最终可包括网格上的所有的信息服务资源，从而提供实时、动态、有效的资源信息供资源调度算法使用。同时，P2P 作为一种分布计算模型，可以很好地解决集中式资源调度方法无法避免的数据不一致和性能瓶颈等问题。

我们认为，在高度动态的环境中进行资源调度，把任务请求发布出去的方式比收集资源信息的方式更加有效，因为收集来的资源信息会很快过时，利用过时的资源信息进行资源调度是毫无意义的。基于此思想，我们设计了一个基于对等网络资源发现的网格资源调度算法 GChord。该算法把资源调度问题转化为资源发现问题，网格结点的可利用资源信息以服务的形式发布，任务对资源的需求也以服务的形式发布，GChord 算法使用扩展的 Chord 算法将任务需求在网格中转发，然后通过资源发现的方式将任务映射到可以提供满足要求的资源服务的结点上。

1. Chord 协议

麻省理工学院设计了一种分布式的可扩展的查找和路由协议 Chord，Chord 通过将(key，value)对中的 key(如文件名)和网络结点分别进行哈希，并将哈希值映射在相同的值空间，将(key，value)对存储在最接近 key 的哈希值的结点上。一般来说，Chord 采用一致性哈希方法将 n 个结点哈希到具有 $\log n$ 位的标识环上。每个结点 x 存储指向它的直接后续 Successor(沿环顺时针方向的最近结点)。它也维护一个有 $\log n$ 个表项的指针表(finger table)。第 i 表项存储 $x+2i-1$ 的后续标识。Chord 路由算法采用了类似二分查找的方法，每次查找发送的消息数为 $O(\log n)$。稳定状态下，在一个 n 结点的系统中，每个结点需要维持 $O(\log n)$ 个其他结点信息，解析查找需要 $O(\log n)$ 个消息。当结点加入和离开系统时，Chord 为维持路由信息一致性最多发送 $O(\log^2 n)$ 个消息。当路由信息不一致时，性能有限降低。在结点的加入和离开比较频繁的 P2P 网络中，每个结点只需要更新少量信息就可保证正确的路由查询。为了容错，每个结点可维护一个后续链代替单个后续。Chord 协议简单，具有可证明的协议正确性和良好的性能。

2. 基于 Chord 的网格资源调度模型

针对网格环境的动态性，如资源结点的频繁加入和离开、资源信息属性的频繁变化等，我们将网格资源组织在一个结构化的 P2P Overlay 中，设计了一个基于 Chord 协议的网格资源调度模型 GChord。

之所以选择结构化的 Overlay Network，而不使用没有采用非结构 Overlay Network，是因为非结构化 P2P 往往采用洪泛方式进行资源信息的发布和动态更新，会形成很多冗余信息，必将增大通信负载。同时，和其他协议相比，Chord 协议简单、易维护，因此选择采用 Chord 协议来维护 Overlay 上每一个结点的路由表。

本节的研究工作仍然基于第 6 章提出的具有超级结点对等网络 Overlay Network 的网

格资源管理体系结构。GChord 假设网格系统存在多个虚拟组织 VO，每个 VO 使用一个资源管理代理(Resource Management Agent，RMA)来管理 VO 内部的资源信息；不同 VO 的 RMA 组成一个使用 Chord 协议维护的环形结构化 P2P Overlay Network，RMA 之间可以相互通信。图 7-8 描述了 GChord 网格资源调度模型主要研究的问题。

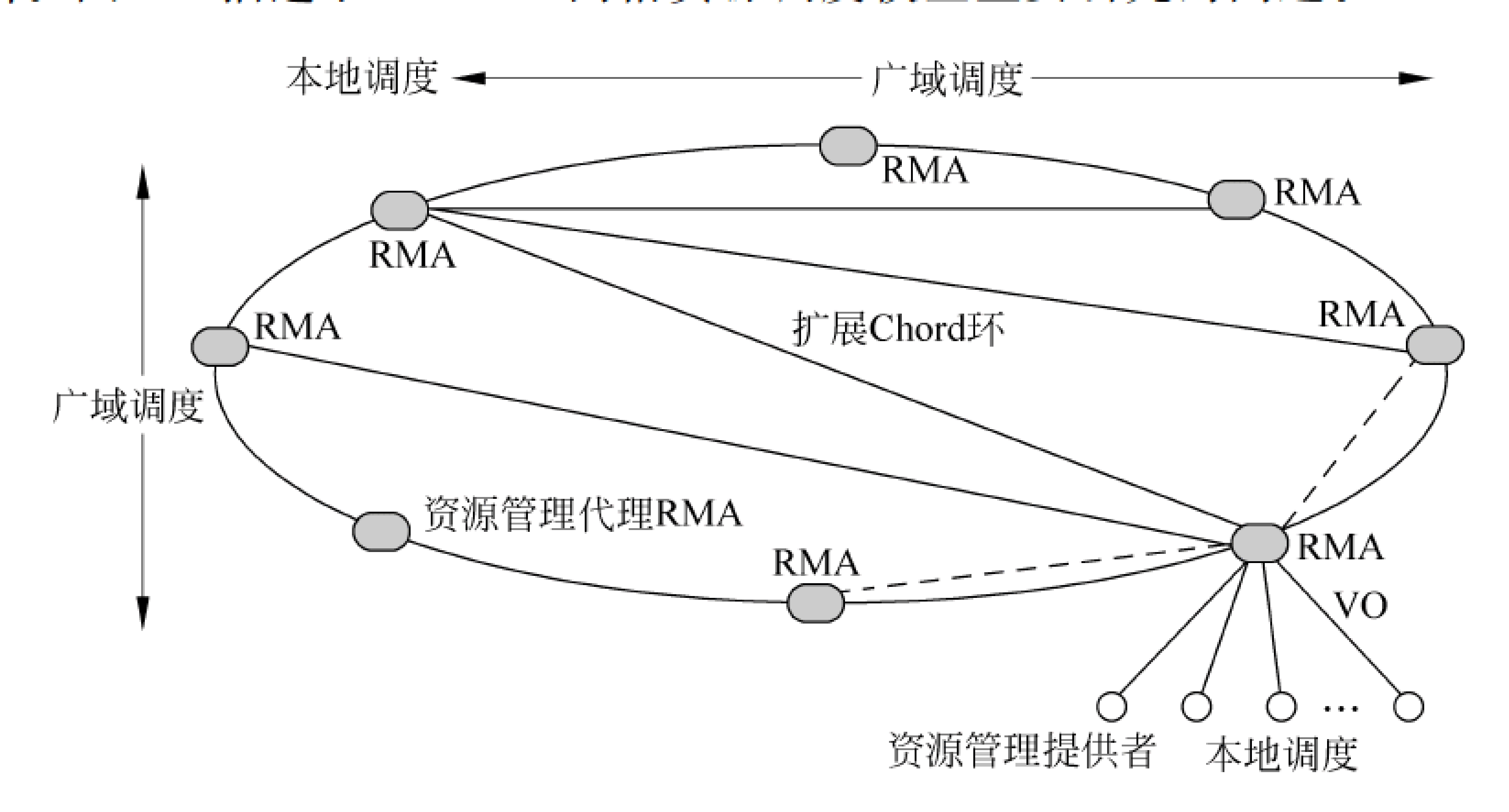


图 7-8　GChord 网格资源调度模型

GChord 将网格环境中的调度分成广域调度和本地调度两种。本地调度指任务被映射到某个计算结点之后，计算结点内部的系统调度，对应图 7-8 中所示资源管理提供者以下层次的调度，这个问题不属于本书的研究范围。本书研究广域调度包括图 7-8 中所示 VO 内部的调度以及由 RMA 组成的扩展 Chord 环上的调度。

在 VO 内部，每个资源管理提供者将其可利用资源信息以服务的形式发布到所在 VO 的 RMA。每个 RMA 负责维护一个信息表和一个路由表，信息表记录 VO 内部的本地资源的属性，路由表用来记录扩展 Chord 环上的其他 RMA 的位置信息。和 Chord 协议不同，RMA 仅维护其 VO 内部的资源信息，并不对其他 RMA 的资源信息进行备份。当资源提供者的资源属性发生变化时，例如，其工作负载发生了变化，该结点使用 push 方式修改 RMA 上的信息表内容即可。

在进行任务调度时，将任务对资源的请求使用 Chord 协议在网格中转发，在查找可用资源结点的过程中，将可利用资源信息和任务请求信息进行匹配，直到找到满足要求的结点为止。然后将任务部署到目标结点，进入目标结点的本地调度范畴，开始计算。图 7-9 和图 7-10 分别描述了 GChord 网格任务调度模型的资源信息发布和资源定位过程。



图 7-9　GChord 模型的资源发布过程　　　图 7-10　GChord 模型的资源定位过程

如何把基于 P2P 的调度模型应用于网格环境，例如，如何将 P2P 的资源查询思想有效地应用于网格任务的调度，是我们要解决的问题。因此，我们在设计 GChord 调度算法时主

要解决了以下三个问题：

(1) 创建一个或多个结点可以注册和加入的虚拟组织；

(2) 以服务的形式发布结点可利用资源信息；

(3) 可利用资源信息的服务发现。

针对以上三个问题，接下来的章节将对分布式资源信息注册器、信息服务的发布格式以及服务发现的方法进行详细阐述。

1) 分布式注册器

每个 RMA 在提供网格服务访问的同时，还是扩展 Chord 环中的一个对等结点。图 7-11 详细描述了 RAM 的各组件。

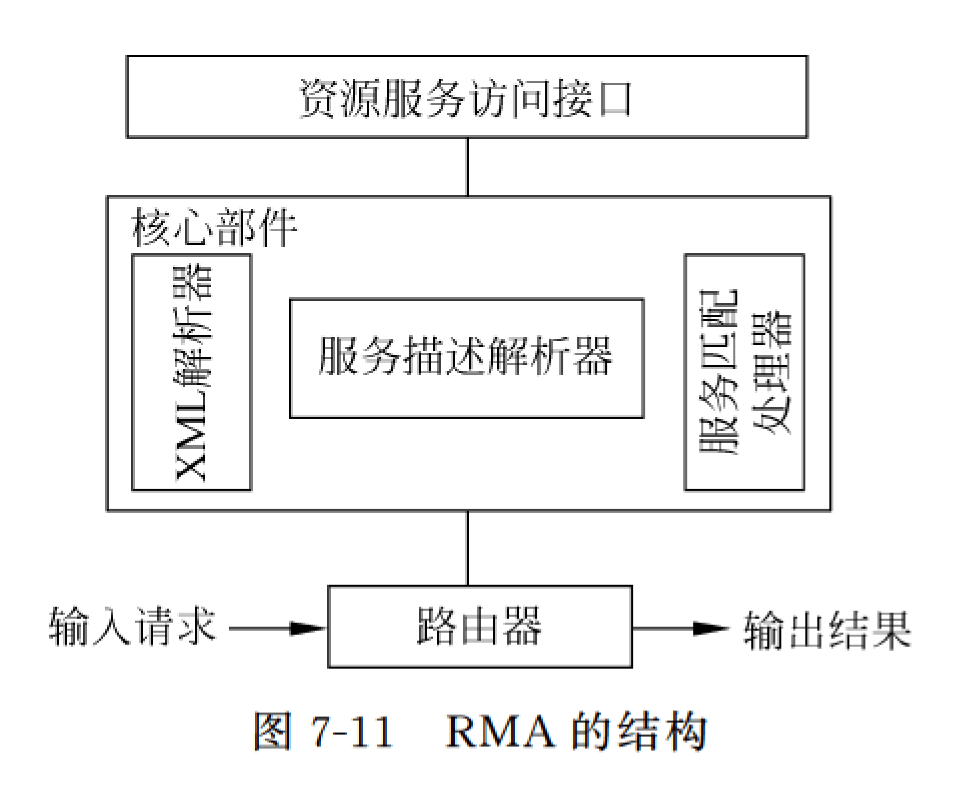
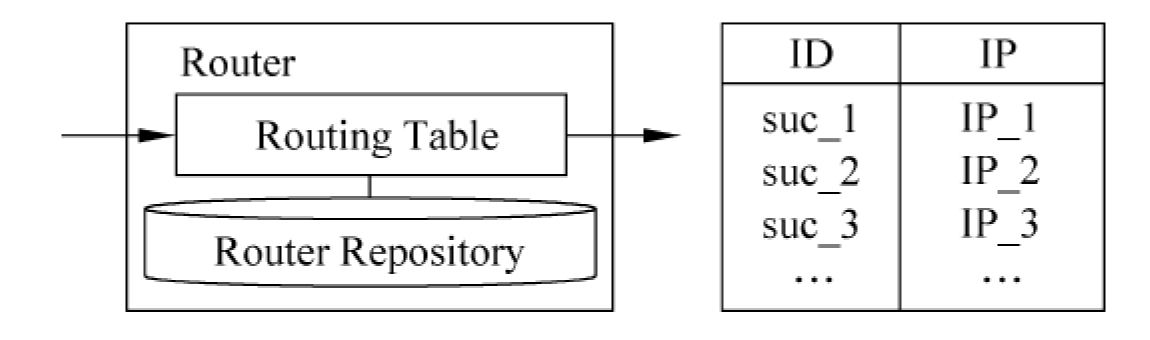


图 7-11　RMA 的结构

每一个 RMA 包含三个部件：资源服务访问接口、核心部件和路由器。资源服务访问接口以服务的形式存在，为结点提供了服务发布和服务发现的接口。核心部件包括一个 XML 解析器、一个服务描述解析器和一个服务匹配处理器。服务匹配处理器是一个判断可利用资源服务是否满足任务需求的匹配算法，判断结果通过资源服务访问接口返回。XML 解析器用来解析资源服务的 XML 表示文档，同时也可以被其他部件使用。路由器把任务请求在 Overlay 网中传递，还负责将路由结果返回给请求结点。路由器由路由表和注册映射表构成，如图 7-12 左半部分所示。路由表的结构和 Chord 路由表完全相同，最大可表示 160 项路由信息，每一项由 160b 的 ID 和 32b 的 IP 地址组成。路由表结构对应如图 7-12 右半部分。

2) 资源服务的发布

为了实现资源的动态调度和负载均衡，资源信息应该不断更新。采取不同的更新策略将大大影响系统的性能。在 GChord 中，每个 RMA 将其可利用资源信息以服务的形式发布在路由器的注册映射表中，如图 7-13 所示。信息更新采取定期更新和系统事件触发的方式进行。图 7-14 形象地描述了在 RMA 注册映射表中存储的资源服务。



ID	IP
suc_1	IP_1
suc_2	IP_2
suc_3	IP_3
…	…

图 7-12　路由器结构和路由表

Service	Description List
Service 1	Description 1
Service 2	Description 2
Service 3	Description 3
…	…

图 7-13　注册映射表

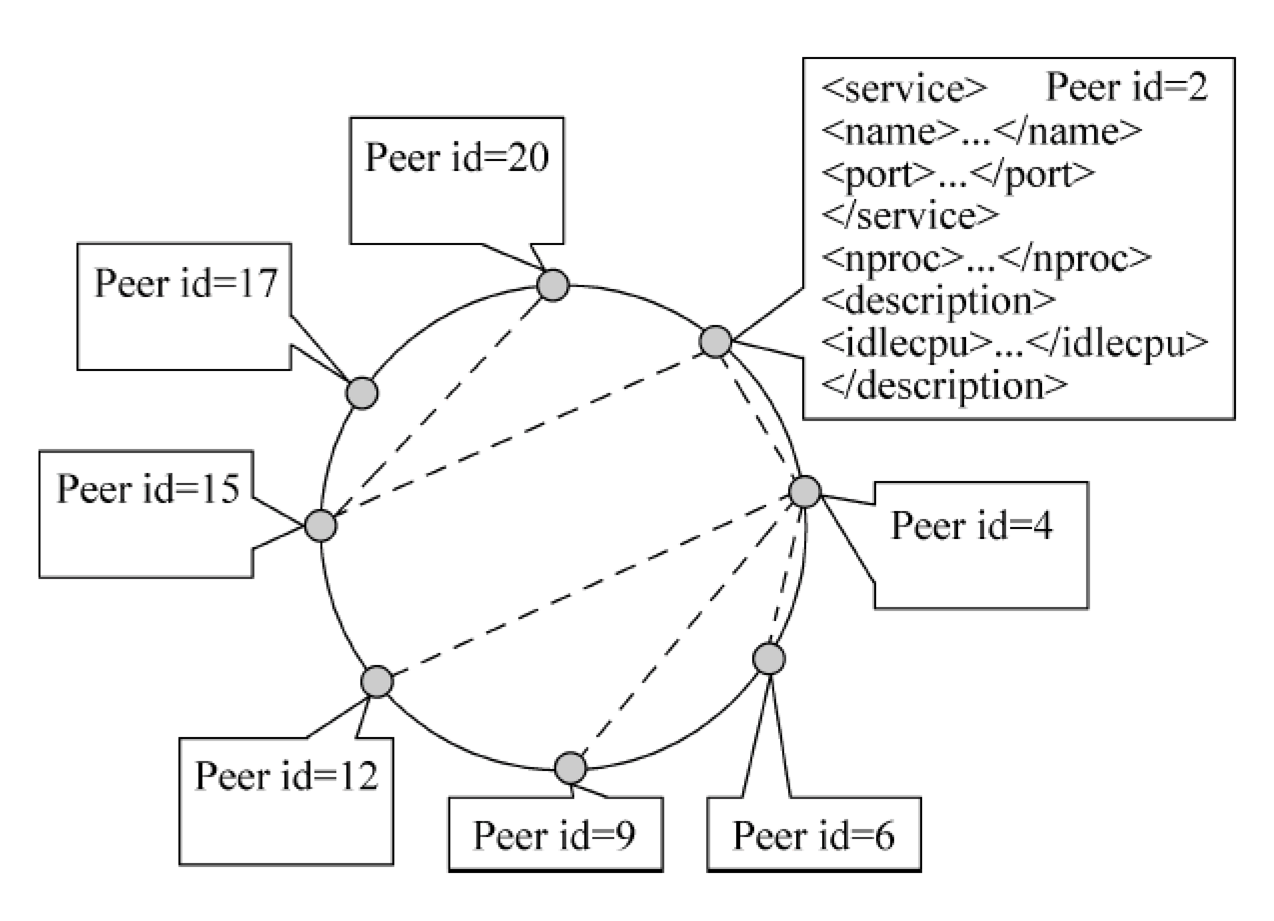


图 7-14　资源服务发布结点

我们使用 WSDL[25] 来描述服务接口。下面代码是一个可利用资源信息服务的例子，服务部分由 WSDL 文档自动生成，描述部分由结点的实时工作负载状况确定。

```
<?xml version = "1.0" encoding = "UTF - 8"?>
< services >
    < service >
        < name > ResourceService </name >
        < documentation >
        Description of the available resource
        </documentation >
        < location > gideon. csis. hku. hk </location >
        < port > 22 </port >
    </service >
</services >
< description >
    < cluster >
        < location > gideon. csis. hku. hk </location >
        < port > 22 </port >
        < type > Gideon300 </type >
        < nproc > 16 </nproc >
        < environment > MPI </environment >
        < environment > pDOCK </environment >
        < idlecpu > 3 </idlecpu >
        < availablememory > 2. 1GB </availablememory >
    </cluster >
</description >
```

使用 location 和 port 来描述如何访问一个资源结点，对结点的硬件模型、包含的处理器个数进行了说明。图 7-15 描述了一个可提供 16 个处理器的 Gideon300 集群，在此集群上可以运行 MPI 和 pDOCK 程序。RMA 使用 idlecpu 和 availablememory 来表示每个网格结点的工作负载状态，匹配算法使用这两项来判断是否将任务分配给此结点进行处理。在动态环境中，

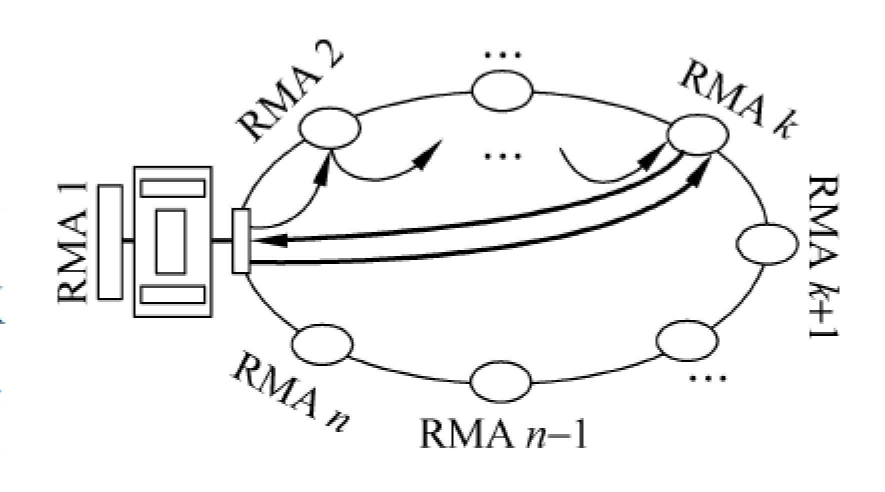


图 7-15　GChord 的资源发现过程

这两项的值容易失效，因此必须经常刷新以确保真实反映网格的实时负载状态。

3）资源服务发现

当 RMA 接收到任务请求时，资源发现进程被触发，此进程的核心是对可利用资源服务和任务请求信息进行匹配。图 7-15 所示为 GChord 资源定位过程，即发现资源服务过程的例子。RMA1 接到任务对资源的请求后，启动资源发现进程。RMA1 首先对自己维护的信息表进行查找，如果不能找到满足请求条件的表项，RMA1 按照 Chord 路由策略将请求发送到 RMA2；RMA2 进一步转发该路由请求，直到找到目标结点 RMA*k*。RMA*k* 结点向将路由结果返回给 RMA1，然后 RMA1 将任务部署到 RMA*k* 结点，任务进入 RMA*k* 结点的排队系统等待处理。

为了找到合适的资源服务，任务请求也使用 XML 的格式进行表述。下面代码是一个任务请求的 XML 描述格式，此请求使用了复合条件。

```
<request><and><requirement>
            <environment>pDOCK</environment>
            <cpuamount>2</cpuamount>
            <memroy>200MB</memroy>
        </requirement>
        <application>
            <name>molecule docking</name>
            <binary>
                <file>binary/docktest</file>
                <inputfile>binary/docktest.infile</inputfile>
            </binary>
            <environment>pDOCK</environment>
</application></and></request>
```

7.2.2 算法描述

下面是 GChord 算法的伪代码。

```
SS: the resource services set;
SS(i).item: the item value of resource service i;
RS: the request services set;
RS(j).item: the item value of request service j;
Chord(m): get the neighboring peer of peer m using Chord algorithm;
Match(SS(i), RS(j)): function to decide whether the requirement of request service j can be
satisfied by resource service i;
Fuction Match(SS(i), RS(j)){
    Parse RS(j).requirement;
    If each RS(j).requirement <= SS(i).item
        return True;     //Matched;
    Else return False;   //UnMatched;
}
GChord {
    For each element in RS(i).requirement{
        If Match(SS(i), RS(i)){
            // Check the local VO resource service firstly;
            Allocate the according task to SS(i).location;
            Update the SS(i).idlecpu;
```

```
        }
        Else {
            j = Chord(i);
            If Match(SS(j), RS(i))
                Exit successfully;
            Else {
                Continue GChord until all the peers have been accessed;
                Exit unsuccessfully;
}}}}
```

GChord 算法的流程图如图 7-16 所示。

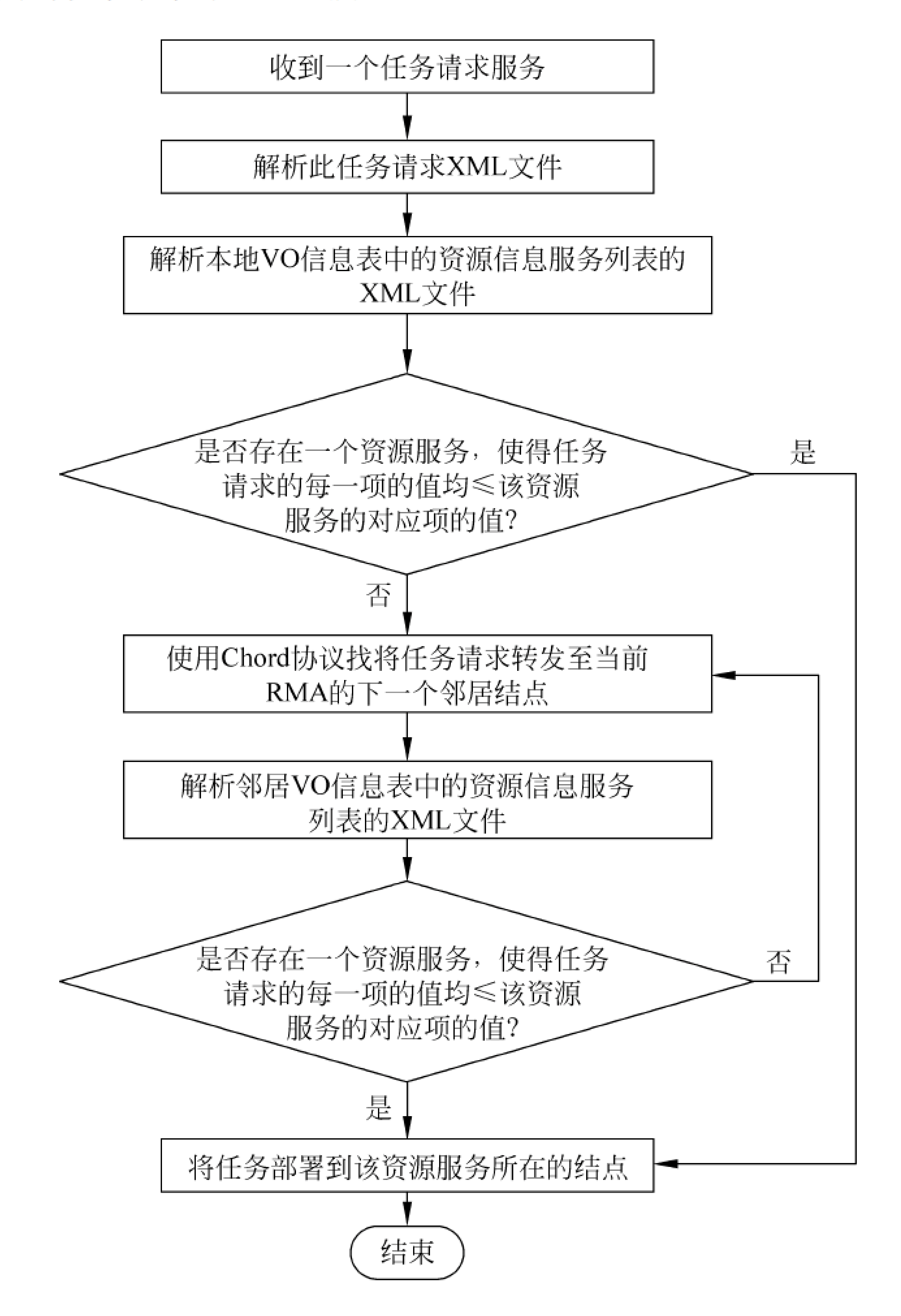


图 7-16　GChord 算法流程图

GChord 算法在进行符合任务需求的资源服务发现过程中，会首先检查本地 VO 中是否存在符合任务执行条件的资源提供者。如果答案是肯定的，那么任务将直接在本地处理；反之，RMA 会根据 Chord 路由表中的路由信息将此任务请求转发给其他的邻居 VO。这样设计的原因是基于以下考虑：

GChord 算法的资源发现过程并不是为了给每一个任务请求寻找最佳的解决方案，而

是致力于在最短的时间内寻找一个最临近的可利用的网格资源提供者。虽然这可能会影响到网格系统全局的负载均衡效果,但是由于网格用户都希望找到一个物理位置尽量接近本地的资源提供者,以此来降低网络通信和任务数据、计算结果的传输开销。GChord 算法的设计方式就可以尽量降低路由开销和通信开销,因此其是可行的。

7.2.3 理论分析

本节对 GChord 的路由时延和空间负载特性做了理论上的分析,结果表明,GChord 的路由时延和空间负载是合理的。

1. 路由时延

我们使用在扩展 Chord 环中的逻辑路由跳数来分析算法的路由时延。因为 Chord 算法的逻辑跳数和结点个数 n 存在 $\log n$ 的关系,GChord 算法的路由表和 Chord 算法的路由表相同,因此 GChord 算法的逻辑路由跳数和结点个数,即 VO 个数或者 RMA 个数,之间也存在 $\log n$ 的关系,即路由时延随着结点个数的增加呈指数增加。如果单个逻辑跳的时延是 k,则全局时延为 $k\times\log n$,对网格系统而言,该时延是合理的。

2. 空间耗费

GChord 的空间耗费包含两部分:路由表的空间耗费和注册映射表的空间耗费。为了便于分析,首先给出一个定义和两个假设。

定义 7-6 *存储单元。把路由 ID 和相应 IP 的存储大小计为一个存储单元,大小用 σ 表示。*

假设 7-1 *RMAs 的 ID 在 ID 空间内均匀分布,网格系统中共有 n 个 RMA,平均每个 RMA 在其路由器的注册映射表中发布 m 个可用资源信息。*

假设 7-2 *每项可用资源信息的平均空间耗费为 $k\times\sigma$。*

因为路由表中共有 $\log n$ 项,每一项的空间耗费为 σ、192b,故路由表的空间耗费为 $\log n\times\sigma$。根据定义 7-6 和假设 7-1、假设 7-2,每个注册映射表有 m 项注册信息,因此注册映射表的空间耗费为 $m\times k\times\sigma$。故 GChord 总的空间耗费为

$$\text{Space}_{\text{GChord}} = \log n\times\sigma + m\times k\times\sigma \tag{7-1}$$

为了说明此基于结构化 P2P 的分布式网格资源注册器的空间耗费是合理的,下面分析集中式注册器 UDDI 的空间耗费,因为 UDDI 使用一个中央注册器来维持所有的注册信息,故 UDDI 的空间耗费为

$$\text{Space}_{\text{UDDI}} = n\times m\times k\times\sigma \tag{7-2}$$

比较式(7-1)和式(7-2),可以得出结论,GChord 的空间耗费比 UDDI 的空间耗费低得多。

7.2.4 算法实验及结果分析

为了验证 GChord 调度算法的性能,基于一个使用 C++实现的网格实验系统,在真实的网格环境中,我们测试了调度算法的整体性能、负载均衡特性和鲁棒性;并且将使用 GChord 算法和并行方式下的任务执行时间做了比较。

1. 实验环境

网格实验系统由分布在 Internet 上的四台集群和通过 LAN 连接的四台 PC 机构成,结构如图 7-17 所示。

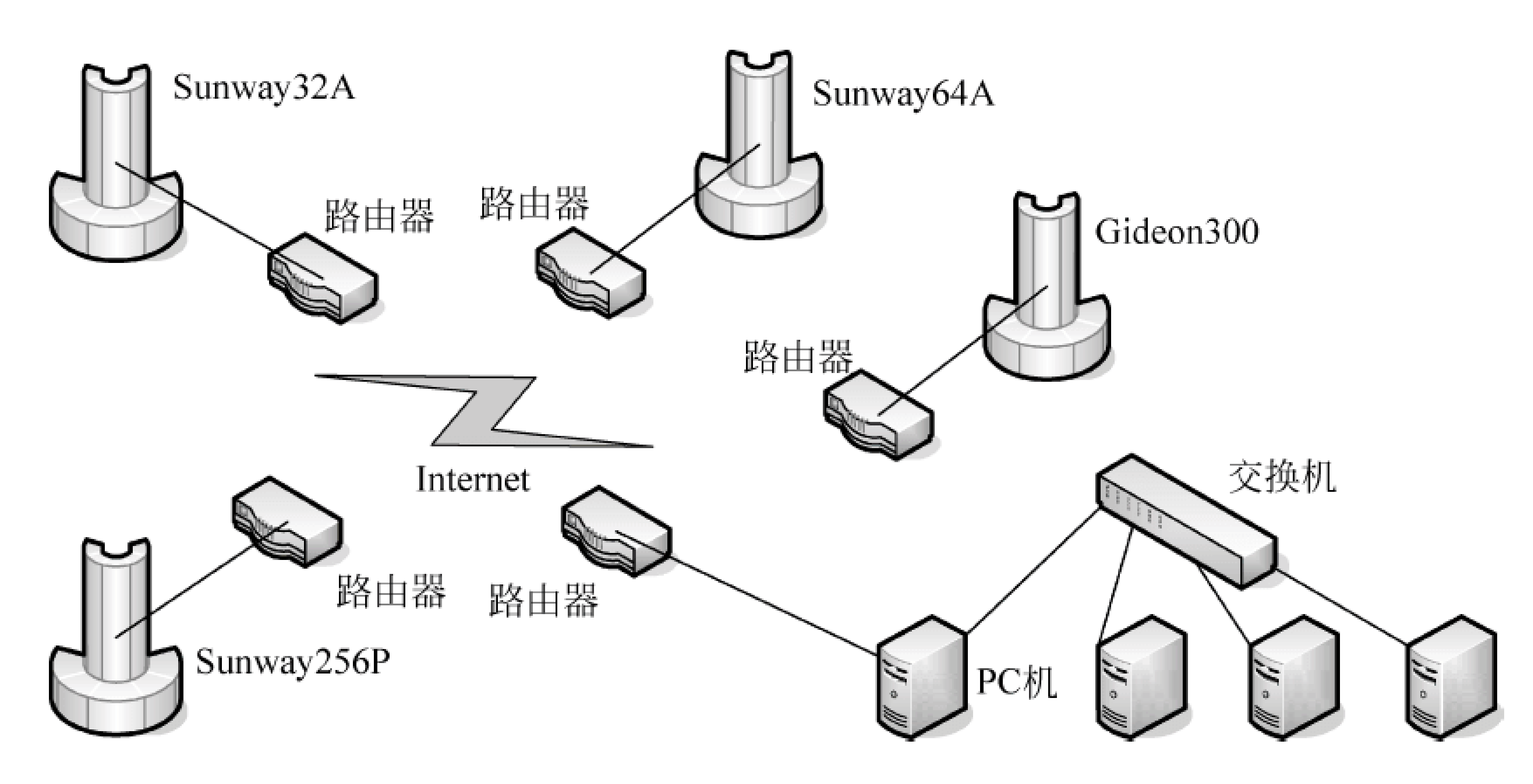


图 7-17　网格实验系统结构图

使用本节扩展的 Chord 协议,将网格中的异构计算结点:4 台集群和 4 台 PC 机,通过高速以太网组织在一个超级结点对等网络中。每个网格结点的详细配置信息如表 7-3 所示。集群包括 3 台神威(Sunway)机群和 1 台 Gideon300 PC 机群,其中,3 台神威机群分别含有 32、64、256 个计算结点,分别选取 8 颗、8 颗、32 颗 CPU 参与网格计算,同时选取 Gideon300 上的 16 颗 CPU 参与网格计算。4 台 PC 机的配置为:1GHz Pentium 4 处理器,256MB RAM,40GB 硬盘。

表 7-3　网格实验环境结点配置表

主机名	位置	选用 CPU 数	CPU 类型及主频	内存	网　络	Dock Benchmark 值
Sunway32A	无锡	8	Alpha/833MHz	512MB	Myrinet	4.33
Sunway64A	无锡	8	Alpha/833MHz	512MB	Myrinet	5.43
Sunway256P	北京	32	Alpha/2.0GHz	1GB	1000Mb	9.73
Gideon300	香港	16	Pentium4 2GHz	512MB	Fast-Ethernet	7.63
PCs	上海	1	Pentium4 1GHz	256MB	100Mb/s Ethernet	0.53

定义了 3 个虚拟组织:2 台位于无锡的神威集群属于 VO1,位于北京的神威集群和香港的 Gideon300 集群属于 VO2,4 台 PC 机属于 VO3。每个 VO 内部的计算结点都可以将自己的可用资源服务信息发布到负责管理此 VO 的 RMA;同时,RMA 提供资源访问接口,通过 RMA 之间的相互通信进行服务发现,从而实现网格任务的调度。

在此实验过程中,网格任务为蛋白质分子的 DOCK 实验,即计算一个大的蛋白质分子结构和商业分子数据库中的分子结构的匹配程度。蛋白质分子的 DOCK 实验具有高度的并行性,并行化后的子任务之间不需要通信,非常适合在网格系统中进行计算。实验过程中将大的蛋白质分子和 Specs 数据库中的样本分子进行结构匹配。

在进行 DOCK 实验过程中,使用 GChord 算法对实验系统中可利用的网格计算资源进行调度。算法的 RMA 信息表的资源服务信息主要包括所辖 VO 内部计算结点的空闲 CPU 数目和可利用存储空间。空闲 CPU 数目使用实时测得的系统 benchmark 值表示,量化为当前可利用计算单元的个数。此 benchmark 值在实验系统初始化时确定,并且保持固定不变。实验中考虑了系统进行最初的任务并行化所需的时间和最后的任务计算结束后的

结果收集时间，这两个过程实质上为各计算结点和主进程所在结点之间的通信开销，可以作为任务对资源请求的一部分来处理，即使用单位 CPU 数量来表示。

2. 实验结果

1）整体性能

我们使用任务总体执行时间 makespan 来评价算法的整体调度性能。为了分析算法的性能，将实验结果和并行方式下的任务总执行时间进行了比较。具体做法是：对于同一个 DOCK 实验，首先在一台集群上进行并行计算，记录从任务提交到任务完成所需计算的时间。然后，将同一个任务提交到网格实验系统中，使用 GChord 算法进行任务调度，记录任务的计算时间。最后，在分析计算结果同样有效的情况下，比较两种不同的调度模式下任务的计算时间。改变任务的输入，提交不同的大分子，或者选择 Specs 数据库中不同的样本分子，重复上述过程，进行多组实验。图 7-18 记录了将同一个大分子和 Specs 数据库中的 1000、5000、10000 个样本分子进行 DOCK 匹配计算，分别在 Sunway32A、Sunway256P 上进行并行计算和在网格实验系统上使用 GChord 调度算法的情况下测得的任务执行时间。每种情况的实验数据是同样条件下重复 5 次的平均结果。

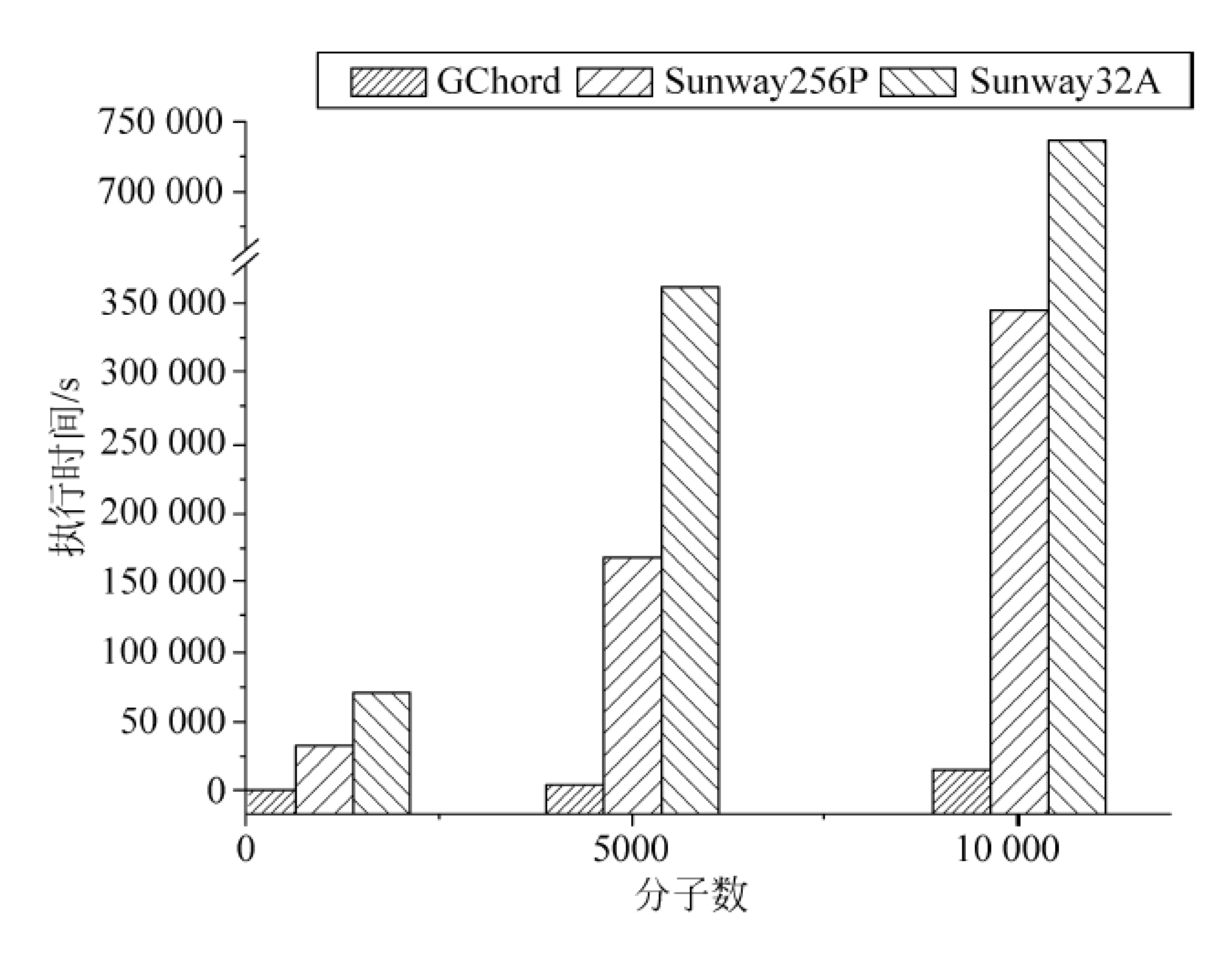


图 7-18　不同环境中任务的执行时间

从图 7-22 可以看出，得到同样有效的计算结果的条件下，选择 1000 个样本分子进行 DOCK，在并行方式下，需要在 Sunway32A 上运行 71 398. 12s，在 Sunway256P 上运行 33 355. 65s，而在网格实验系统上使用 GChord 调度算法，仅需要 1193s 的计算时间。并且，随着选择的 Specs 数据库中样本分子的数量增多，例如，样本分子数量为 10 000 时，在 Sunway32A 上运行 737 591. 12s，在 Sunway256P 上运行 168 143. 25s，网格实验系统需要 16 194s，网格实验系统呈现出很好的性能。

由此分析可以得出结论：网格实验系统聚合了 Internet 上的计算能力，并且通过各个网格计算结点间的相互协作，大大缩短了蛋白质分子 DOCK 实验的计算时间，进一步说明了 GChord 算法的整体性能很好。样本分子数量越多，计算时间越短，说明了由于网格结点间通过 Chord 网络，建立了 P2P 的协作关系，可以加速计算的进程。

2）负载均衡

GChord 在进行资源分配时，根据存储在 RMA 信息表中的计算结点的实时资源属性进行网格任务调度。能否实现网格结点间的负载均衡，获得全网格的负载均衡是评价动态资源调度算法的一个重要测度。为了评价 GChord 算法的动态调度性能，本部分的实验采用记录分配给每个网格计算结点的任务数量的方法测量结点的负载状态，以此来评价 GChord 算法的负载均衡特性。

取 Sunway32A 和 Sunway256P 作为进行实验的网格结点，因为这两台集群在参与网格计算的 CPU 数量、空闲程度上有较大的区别，利于分析实验结果如表 7-3 所示。实验中，向网格实验系统提交一个蛋白质大分子，选取 Specs 数据库中一定数量的样本分子进行 DOCK 计算，记录 GChord 的资源调度结果，即分配给每个计算结点的分子数目以及此结点当前空闲 CPU 数量，然后计算任务分配比率。重复此实验，改变样本分子数量，记录调度结果。表 7-4 是分别选取 1000、5000、10 000 个样本分子时的实验结果。分析图中的数据，可以看到任务分配基本反映了实验系统中空闲计算资源的分布情况，计算资源是根据结点的计算能力和实时工作负载状态进行调度的。实验数据表明 GChord 算法很好地实现了网格实验系统的负载均衡功能。

表 7-4　网格实验系统的负载均衡状态

分子库大小	Sunway32A 计算任务量	Sunway256P 计算任务量	总计算时间/s
1000	108（10.8%）	892（89.2%）	1193
5000	604（12.1%）	4396（87.9%）	5349
10 000	968（9.7%）	9032（90.3%）	16194

3）鲁棒性

GChord 采用基于超级结点对等网络来组织网格 Overlay 网络拓扑以期达到提高网格实验系统的鲁棒性的目的。这里设计了实验来测试该设计的有效性，随机断开几个网格结点，测试系统重新恢复稳定的能力。在实验中，使用 4 台 PC 进行测试，将其他 3 台集群手动屏蔽。这样做的原因有以下两方面：

（1）缩短实验周期。因为 4 台 PC 使用 LAN 连接，不使用 Internet 上的通信链路，可以大大减少网格系统的通信开销。

（2）排除集群自身资源管理软件自恢复功能带来的干扰因素。鲁棒性实验的目的是测试 GChord 算法的 P2P 特性，即一个结点失效后，P2P 网络中的其他结点是否可以弥补此结点失效带来的损失。该过程对于用户是透明的。而集群上安装的本地资源管理软件，例如 LSF 和 PBS 具有系统故障恢复功能，在断电、网络故障等情况下可以在一定程度上将意外中断的任务恢复。这有可能影响我们对实验结果的分析。

实验的具体做法是：从 Specs 数据库中选择 1000 个样本分子和一个大的蛋白质分子进行 DOCK 计算，使用 GChord 算法进行调度，观测计算返回的结果以及计算耗费的时间。待计算完成任务总量的 10%后，随机断开一定比例的计算结点与实验系统的连接，模拟网格通信链路故障，待网格系统自动重新恢复稳定状态后，等待系统返回完整的计算结果，记录计算所需的总时间，分析计算结果的有效性。

实验结果显示，过了 1854s 后，网格实验系统返回了正确的计算结果。分析其原因，因

为 GChord 的资源管理方式是 P2P 模式，使用 Chord 协议来维护结点间的连接，即使一个结点失效了，任务能够传递给 P2P Overlay 网络中的其他结点继续执行。也就是说，实验系统不存在明显的应用调度失败。由此可以得出结论，GChord 算法具有很好的容错性和高的鲁棒性，部分计算结点的异常不会影响系统的整体性能。

虽然发生结点失效后，正确、完整的计算结果返回的时间比正常计算需要的时间稍长（实验测得的正常时间为 1193s），但重要的是，系统可以返回正确的计算结果，这一点对于需要确保 DOCK 实验结果正确性的网格用户而言至关重要。

上面介绍了基于结构化 P2P 的网格资源管理和调度算法 GChord，该算法具有自组织、可扩展和自适应等优点；通过资源发现的方式进行网格任务的动态调度，解决了采用信息收集方式进行资源调度的方法存在的信息过时、数据不一致的问题；GChord 对 Chord 路由算法进行扩充，使扩充后的算法能更加适用于网格资源的动态性；算法实现过程中主要解决了网格 Overlay 网络的拓扑组织形式、资源的发布方法、资源的更新、资源发现以及路由算法等问题；通过大量的实验证明了该方法的有效性。

7.3 基于多代理协同计算的负载均衡算法研究

负载均衡是网格资源调度的一个重要研究问题，分布式系统因为其物理分布的分散性和结点的自治性，使得研究全局负载均衡问题比较困难。经研究发现，多代理系统具有自学习、自组织的特性，利用多代理的分布性和协同性，通过代理对未知部分知识的学习，可以进行分布式系统的全局、动态的资源调度，因此，本节在研究了多代理系统的基础上，分析了多代理系统可以应用于网格系统的原因，然后设计了 rwAgent 算法。该算法基于多代理协同计算理论，通过代理间的通信、协同、自组织和调度管理，能较好地解决网格资源的动态调度问题，使得网格系统可以获得全局的负载均衡状态，从而具有较高的吞吐量。通过建立数学模型对 rwAgent 算法进行了理论分析，最后使用实验验证了算法的可行性。

7.3.1 多代理协同计算

负载均衡[26~32]一直是分布式系统的研究热点，同样也是网格资源调度中的一个重要研究问题。如果一个网格系统在处理任务时可以始终维持良好的负载均衡的状态，那么该网格系统就可以获得高的吞吐量，从而显著提高网格系统的性能。

目前很多负载均衡技术的研究主要用于基于集中控制的系统或者集群内部等，一般无法适用于分布的、大规模的、动态的、缺乏集中控制的环境。因此本节将对网格环境中的负载均衡问题进行研究，以期达到对网格资源进行动态调度的同时，实现网格系统全局负载均衡的目的。本书在负载均衡方面的研究仍然是基于广域范围的，即网格的全局负载均衡状态，而网格结点内部的负载均衡，例如集群内部的负载均衡问题，不属于本书的研究范围。

针对现有的一些致力于解决分布式系统的全局负载均衡问题的研究，根据实现效果可以分为动态负载均衡和静态负载均衡两种方式，根据实现方式可以集中式调度和分布式调度两种方式。

集中式调度实现的负载均衡，被很多基于 Globus Toolkit 的网格系统广泛采用，例如

Condor-G[33] 和 Nimrod/G[34]，但是这种集中式调度的方法，容易引起单点失效和性能瓶颈问题，动态调度性能不好，不适用于网格结点频繁加入和退出的情况。此外，这些调度方式均不提供 QoS 的保证。

分布式调度往往采用管道通信的方式，例如 BMRB[35] 和 BLAST[36] 系统，但是管道方式很难解决多个资源域、多个虚拟组织之间的负载均衡，因为不同资源域内的管道设定可能是不同的。如果一个分布式系统要实现多个资源域之间的动态的负载均衡，那么该系统必须能够正确选取可以进行通信的基于资源域设定的管道。不同管道间的通信在实现上困难很大，现有的系统未能实现这一点，所以，如果系统用户希望进行多个资源域间的负载均衡，只能通过静态方式实现。因此，管道方式不适合进行系统的动态负载均衡调度。

分布式系统的全局负载均衡因为结点分布的物理分散性以及结点的自治性，使得研究其负载均衡问题比较困难。网格系统、对等网络等均由于结点的动态性而不能使用静态管理的方式达到负载均衡的目的。

经大量的研究、分析，发现多代理系统具有自学习、自组织的特性，利用多代理的分布性和协同性，通过代理对未知部分知识的学习，可以进行分布式系统的全局、动态的资源调度。

多代理系统是人工智能的主要分支，它的主要任务是研究有多个代理存在的复杂系统的构造原理及独立的代理行为的协同机制。多代理系统技术是人工智能技术的一次飞跃。由于多代理系统的复杂性，更多研究集中在使用机器学习的方法上。因为多代理系统具有并行性、鲁棒性及可扩展性，受到越来越多的研究者的重视。

1．定义

与单代理系统不同，在多代理系统中代理相互间存在交互。从单个代理的角度来看，多代理系统环境是动态的，代理的行为受到其他代理的影响。多代理之间通过通信，可以开发出新的规则与复杂问题的求解方法，使得在具有不完全知识的情况下处理问题成为可能。同时多代理间通过协同，可以提高单代理的问题求解能力。多代理系统具有很好的协同性、分布性，并具有较强的学习能力、推理能力与自组织能力，因此可以大大提高问题的求解效率。

按照代理间的异构度及代理间的通信，Stone[37] 将多代理系统分为四种：异构无通信、异构通信、同构无通信及同构通信。图 7-19 描述了一个多代理系统。首先系统将要求解的问题分解成多个相互自治的部分，每个部分都有自己的目的，各部分之间可以进行通信。代理的心智集合使用一个抽象模型来定义，该模型对代理进行定义、描述了代理之间的相互作用以及如何组织这些代理。此外，多代理系统还需要定义一系列的结构和机制来描述当网络发生变化时，代理间的组织关系的复杂变化。

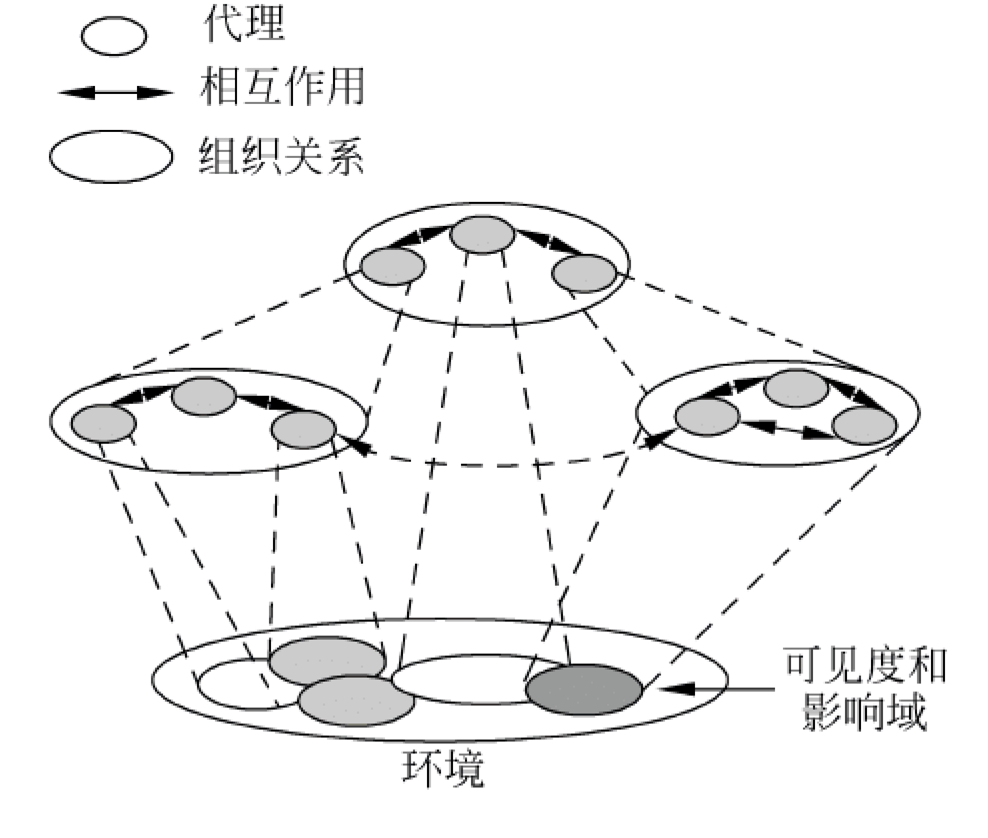


图 7-19　多代理系统的规范视图

2．组织结构

在一个开放的环境中，系统中的代理可以动态地加入或退出系统。因此，多代理的组织机制与多代理之间的相互作用直接相关。多代理组

织通过对代理的行为申明和授权关系给多代理之间相互作用提供了框架。按照 Sycara 的观点，多代理组织是对多代理结构的概念化，是多代理之间的信息和控制关系的模式表示，是在多代理之间问题求解能力的分配。代理的能力和它们在总的决策结构中的组织形式决定了整体系统的行为。通过将具有不同求解能力、观察能力和行为特性的代理用组织结构连接起来，可以使代理的求解行为在组织结构内具有最大的自然性，不必进行更多的规划和通信来进行协调。

多代理组织中的代理以何种方式相互连接，是多代理组织形式需要解决的问题。目前多代理系统的组织结构主要可以分为三种：完全集中式、完全分布式、集中与分布相结合的方式。

在完全集中式的多代理系统中，协调方法是由主控代理完全扩展从属者的行为，由主控代理内部的规划器保证各从属代理的行为彼此协调。集中式协调方法降低了系统的复杂性，减少了用于协商的通信开销，但却要求中心规划器有较强的处理能力，要处理各种可能的冲突，形成一个全局一致的方案，不适合动态、开放的环境，因而在多代理系统中应用较少。

在完全分布式的多代理系统中，各代理处于平等的地位，彼此之间行为的协调通过各代理内部的推理机制或代理之间的多次交互，必要时进行协商实现。这种多代理系统最灵活，但是实现上比较困难，而且代理之间多次的交互协商降低了系统的效率。

在集中与分布相结合的多代理系统中，系统中的代理组成层次结构，上层的监控代理对下层的受控代理有部分的控制能力，在上下层之间有三种控制关系：

(1) 监控代理产生一个完整的规划并发给受控代理，这同完全集中协调方法。

(2) 监控代理只生成部分规划并发给受控代理。

(3) 监控代理不生成规划，只是发给接收者一个高层目标。

与单纯的集中式或分布式的协调方式相比，集中与分布相结合的协调方式的系统集成了前两者方式的优点，既具有相当的灵活性，又具有一定的效率，而且在实现上也比较容易。

多代理的组织行为是通过单代理有规律的行为来体现的。在条件改变时，代理能够按其理性进行调整，保持组织的结构性和自然性，使整个组织有充分的灵活性以应付动态环境。组织重构是指当环境条件发生变化时，或系统成员组成变化时，对组织结构进行调整，使之适应新情况。

组织重构分为宏观和微观两种，组织自设计[38,39]是对系统宏观结构进行动态重构的方法之一，使用重构原语来动态改变系统的宏观结构，这种重构由系统外部触发，非自动进行。由内部触发的重构过程[40]采用基于产生式的自组织分布式代理作为微观结构，代理不断检测系统性能，发现有必要重组时则执行重组原语。

现阶段的多代理系统在组织的适应性方面的研究工作还比较有限，一般只是沿单一的纬度，如最少部分约束等进行。

3. 多代理协同

多代理协同的研究工作始于 Actors 模型[38]，然后出现了合同网协议。合同网协议被认为是关于通信、多代理协同研究的经典工作。在开放的多代理系统环境中，由于代理知识的不完全性，多代理之间需要协同交互才能有效地合作。多代理间的协同方法主要包括：

联合意图模型、共享规划模型、GPGP 框架模型等。

由 Cohen 和 Levesque 等提出的联合意图模型是最著名的理性代理合作理论。联合意图的定义通过单代理的个体承诺来定义。如果代理相信其目标目前没有实现，并有意图实现此目标，它将一直保留这样的目标，除非它相信此目标已实现，或者此目标永远不能实现，或者此目标已与目前情形无关时，该代理才放弃该目标。

共享规划模型规定，多代理之间有一个共享规划，即有一个共同目的。两个代理共同相信：单个代理在适当的时间能够执行某些活动(Activity)；对每个活动，代理存在执行此活动的一个方案(Recipe)；每个代理都有打算(Intend to)在适当的时间去执行它可以执行的特定活动。代理不会采用与联合活动相冲突的意图，代理与其他合作代理通过通信交互的方式对规划的意图进行交互，从而产生对合作活动的行为意图。

GPGP 框架模型(Generic Partial Global Planning)的基本思想是每个代理都构造自己的局部活动视图，视图包含了活动之间的关系以及代理打算去执行的活动。此视图可以不断增长。因为代理之间会经常交换它们活动执行的结果信息，因此，此视图不是完全局部的，而是部分全局的。

在以上几种代理的协同方式的基础上，还产生了许多适应不同应用的协同方式，例如根据游戏理论解决电子商务中的多代理协商问题，通过社会法律来指导多代理协同等。这些协同方式中，都从不同的角度对代理的协同提出了不同的解决方案。

4. 应用领域

通过代理间的通信、协同、自组织及调度管理，多代理系统模型已经在实际中得到了很好的应用。例如在交通控制、智能机器人等方面，可以通过多代理间的协同机制来消解相互间的冲突，从而最大限度地实现代理间的合作；在决策支持系统中应用多代理技术，出现了基于多代理协同的分布式智能决策系统；在网络自动化与智能管理方面，通过定义不同的代理，构成网络中的多代理系统，利用代理的智能，对网络进行主动分析与监测，从而有效地提高网络的效率。在柔性制造系统中，每个加工单元可以看成一个独立的代理，从而构成一个多代理系统，通过代理间的交互来完成生产任务的调度。此外，多代理技术还被应用于信息检索、多处理器的协同设计、分布式计算、普适计算(Ubiquitous Computing)等各个领域[41]。

7.3.2　多代理技术在网格中的应用

多代理技术已经在分布式计算领域中得到了很好的应用，例如，在对等网络中，代理位于对等的计算机上，相互之间可以进行各种信息的来回通信；代理还可以代表其他的对等系统发起一项任务，例如，代理可以被用来区分网络上的任务的优先次序、改变流量流向、在本地搜索文件、确定异常行为如病毒并且在其影响网络前终止该行为等。

Ian Foster 和 Cal Kesselman 等人认为随着多代理技术和网格计算的迅速发展，两个研究领域出现了相互需要对方领域技术的趋势[42]。网格计算呈现出动态、自治特点，需要使用更加灵活、分布式的决策能力来有效地实现管理。另一方面，多代理技术则需要一个鲁棒性好的分布式计算平台来支撑它们之间的通信、协作、制定决策、组织重构和有效管理。因此，多代理技术和网格计算应该相互融合，结合各自领域的优势，解决自身难以解决的问题，将两个领域的研究融合在一起。

将多代理技术应用于网格计算，这方面的研究已经取得了一些成果，例如利用多代理技术进行网格中的资源选择[43]，把资源分配给任务时使用了基于增强学习的多代理技术。代理，即用来制定任务分配策略的逻辑，利用了网格系统的状态监测、资源发现和任务传递等功能。另一方面，通过代理的收集作用，实现了网格系统的全局资源管理。另外，参考文献[44]利用多代理的自动协商技术(特别适用于多种形式的拍卖方)进行网格系统的资源分配，设计者对商品市场和 Vickery 拍卖机制进行了有效性评估。

网格系统和多代理系统在相互融合的过程中可以采用两种方式：层次式(Agent Systems on Top of Grid Mechanisms)和集成式(Integrated Grid/Agent Approach)[42]。层次式的应用相对比较简单，容易实现，即在 Grid 系统的构架上使用多代理技术来解决相关问题。而集成式则需要两种技术深层次的融合，从底层的构建到上层的服务，每个研究点都充分利用两种技术的优势，实现网格技术的多代理化、多代理系统的网格化。

本书采用层次式方法解决网格系统的负载均衡问题，即在网格系统的资源管理体系结构之上，为了达到网格系统的全局负载均衡，在进行资源调度的过程中，借助多代理协同计算技术，实现网格任务的动态资源调度和全网格的负载均衡。

7.3.3 基于多代理协同计算的网格负载均衡算法 rwAgent

基于以上两节对多代理系统的介绍，和对多代理系统可以应用于网格系统的研究分析，本节设计了一个基于多代理协同计算的网格资源调度算法 rwAgent。该算法通过代理间的通信、协同、自组织和调度管理，可以解决网格资源的动态调度和负载均衡问题。

1. 相关工作研究

近几年来，使用多代理技术进行分布式系统的资源调度和负载均衡的研究工作已经取得了一些进展，主要包括利用 Software Agent 进行 P2P 系统和网格系统中的应用研究，例如资源调度、资源发现的算法设计[45]，提供调度 QoS 保证的调度机制研究[46]，基于多代理的网格中间件、网格系统、P2P 系统的设计[47,48]等。参考文献[49]通过实验评价了利用多代理技术进行负载均衡的几种策略。以下就其中几种调度算法做简单的分析。

1) Messor

参考文献[45]中详细描述了 Messor 的工作原理，基于蚂蚁群体(Ant Colonoly)的行为模式，Messor 算法利用一群自治 ant 来均衡网格系统中的负载。在进行网格资源调度时，每个 Messor ant 携带着任务在系统中随机游走，只有在 ant 游走了一段时间(for a while)，并且没有遇到重载结点的情况下，ant 才选择将此任务放下(drop-off)，反之，ant 会继续携带任务游走。Messor ants 的目的是将网格任务尽量分配给不同的网格结点，使得任务在系统中分散分布，而不是将任务堆积在一个结点，以此来保持系统的负载均衡状态。此外，在生命周期内，每个 Messor ant 有两个状态 SearchMax 和 SearchMin。在处于 SearchMax 状态时，ant 试图寻找一个重载结点。如果存在这样一个重载结点，ant 记录此结点的 ID，同时自己的状态变成 SearchMin，继而寻找一个轻载结点。在找到轻载结点后，ant 会使用调度器将重载结点的任务到这个轻载结点，之后，ant 的状态又转变成为 SearchMax，开始下一个寻找重载结点的过程。

2）ARMS

ARMS[48]是将多代理技术应用于网格系统的一个典范，有很多后续的研究工作都基于此系统展开。ARMS 旨在使用多代理技术来提高网格系统的可用性和适应性，ARMS Agents 之间可以进行直接通信，以此来进行资源发布和资源发现，把网格资源分配给不同的网格任务，实现资源调度。ARMS Agents 是同构的，并且被组织在一个层次式结构中，在系统实现过程中，利用了资源性能预测器 PACE。

3）MWP

在 ARMS 的研究基础上，不同的研究小组对多代理在网格中的应用进行了不同程度的改进，例如在参考文献[46]中，Agent 在作为 Service 提供服务的同时，还可以转发服务请求。使用 ASM（Agent Service Manager）实现 Agent 的创建、删除和管理，包括对 Agent 的监控和调度。在创建 Agent 时，给不同的 Agent 赋予一定的优先级，根据优先级的顺序将 Agent 存储在一个堆结构中。在服务发现，即资源调度过程中使用 Agent 的优先级来实现一定程度的 QoS 服务。此文对 ARMS 做了两点改进：第一，利用 Agent 的 Local Knowledge，当同一个服务被请求两次时，第二次的查找时间要比第一次的查找时间短得多；第二，实现了一定程度的负载均衡，Agent 在进行服务查找时，首先在本地资源域进行资源发现，而不是其他资源域。如果本地资源域内不存在这样的服务，或者本地负载过重时，Agent 负责在其他域内进行服务发现。

4）Anthill

Anthill[47]是一个基于 Ant Colonoly 的多代理模拟器，可以用来模拟和分析多代理算法的性能。基于和其他多代理模拟工具包不同，Anthill 是一个 P2P 的系统，其 Java 原型系统基于 Sun Micorsystems 的 JXTA[50]构建，可以提供 P2P 算法的性能分析。另外，Anthill 实现了单个 ant 在模拟状态和实际运行状态的统一配置，因此在发布一个蚂蚁算法时不需要重新对其进行配置。

2. 算法原理

研究表明[49]现有的负载均衡算法，如集中式调度采用的 FIFO 和 Genetic Algorithm，适合于进行本地调度，而多代理技术适合于广域、全局的资源调度。同时，现有的资源发现方式 pull、push 方式分别适用于服务变化较快、请求变化较快的环境，而在服务和请求均呈现高度动态变化的网格环境、P2P 环境中，多代理技术则充分显示了其优点。

本节提出了 rwAgent 算法，将软件多代理的概念引入网格系统的资源调度中，实现网格系统全局的负载均衡。本节的研究工作仍然以第 6 章提出的具有超级结点对等网络的网格资源管理体系结构为研究基础。超级结点组成一个使用 Chord 协议维护的环形结构化 P2P Overlay Network，超级结点之间可以相互通信。

网格系统中，任务执行结点上的工作负载状态是动态变化的，同时也会因为网格结点的频繁加入和离开，使得系统没有固定的网络拓扑。在此动态变化的环境中，很难获得每个结点的实时工作负载状态。因此，集中式的资源调度和负载均衡方法，例如由服务器分配任务给客户端的主从结构不适合应用于网格系统。多代理技术则非常适用于这种动态的、只能获得局部信息的网格环境。因为单个代理会根据网格结点的实际工作负载情况，结合自己的学习经验，通过和其他代理的相互协作，作出决策，从而实现由部分到全局、由已知到未知

的动态资源调度和负载均衡。

rwAgent 算法充分利用 P2P 技术和多代理技术的优点，使用系统中的多个代理间的相互协作来实现网格系统的广域负载均衡。图 7-20 描述了 rwAgent 算法应用的网格环境。图中的代理生成器(Agent Originator)负责生成系统中的所有代理，每一个网格结点都可以是一个代理生成器。

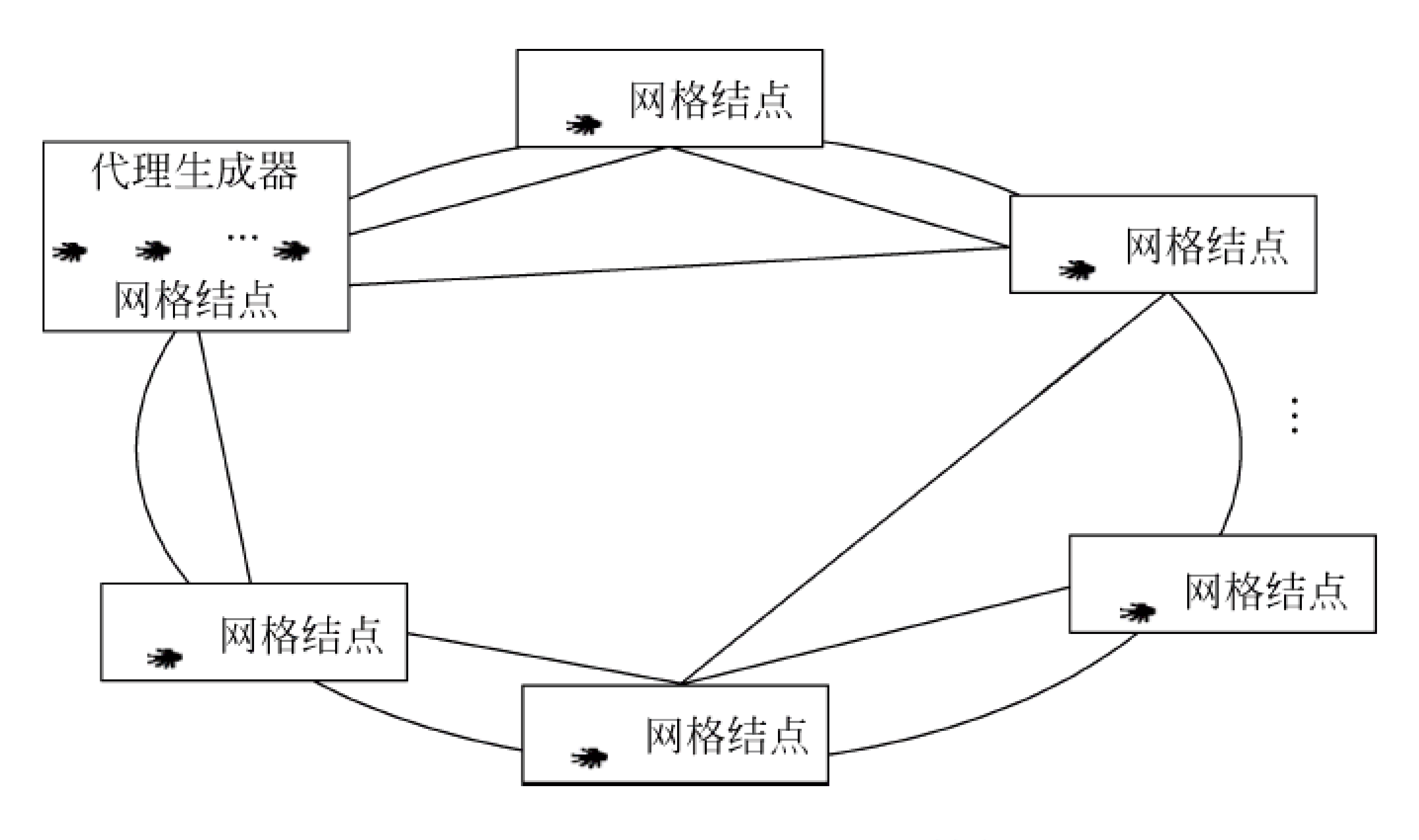


图 7-20　rwAgent 的 P2P 结构

rwAgent 算法的基本思想是：在网格系统将用户提交的任务并行化为多个子任务之后，进行任务并行化的该结点就是一个 Agent Originator，负责生成和子任务同样数量的代理。生成的多代理和系统中的子任务一一对应。每个代理携带一个子任务的任务请求 QoS 在网格中按照 Chord 路由协议移动，即在网格系统 Overlay 层上的各个超级结点之间移动。此时，代理是主动的，在遇到一个处理结点时，可以自由地选择是否将其携带的子任务“放下”，即判断是否将这个子任务分配给此结点处理。如果选择“放下”这个子任务，代理会记录该处理结点的信息(如 Overlay 上的位置或者 IP 地址)，然后返回原始的 Agent Originator；由网格系统将子任务传输给处理结点，子任务将加入该结点的任务排入队列等待处理。如果代理认为某个结点不能满足子任务的 QoS 请求，代理会继续移动，直到找到满足任务请求 QoS 的处理结点为止。

rwAgent 算法规定，代理在遇到某个结点时，根据此结点的实时工作负载状态做出是否将所携带的子任务交给此结点处理的决策。如果此时该结点轻载运行，可以提供满足子任务需求 QoS，代理会做出将此子任务交由此结点处理的决定，将此子任务“放下”，同时，该代理获得一定的正收益，收益的大小根据结点接受子任务后后自身负载的状态确定。反之，如果此时结点重载、或过载运行，代理会选择携带子任务请求 QoS 在 Overlay 网中继续游走，直到遇到可获得正收益的结点为止。从算法的设计可以看出，系统中的每个子任务需求，或者说子任务本身是“移动”的，因此，可以直接把每个子任务看作一个代理。

rwAgent 算法的代理之间并不直接通信，而是驻留在每个超级结点上的知识库里。代理在经过某个结点时做出的调度决策，以及其因此而获得收益都将存储在知识库里。那么当其他代理再经过此结点时，可以根据知识库提供的其他代理的历史信息帮助制定调度决策。

与其他多代理系统不同，rwAgent 算法规定，代理不能将子任务分配给重载、过载运行

的结点,也就是说,代理的收益均为正收益,不存在负收益。这种设计可以保证网格系统处理任务的有效性,不会因为某个结点的过载运行,长时间不返回计算结果,造成任务的再调度,浪费大量的计算资源和存储空间。

判断一个结点的负载状态是轻载还是重载,可以有几种判别方式。第一,可以取决于此结点当前任务排队队列中任务数的多少,如果排队队列的长度很长,称此结点为重载运行,反之,为轻载运行。第二,如果一个任务所需的计算资源数量可以预先测定,那么结点的负载状态也可以根据结点当前可提供的计算资源数量判定。如果结点当前可利用的计算资源数量不足以满足任务的计算要求,则称此结点为重载运行,反之,为轻载运行。除此两点之外,结点是重载还是轻载运行,还和 Agent 最近访问的所有结点的平均负载相关。如果和其他结点相比,当前结点可以提供较多的计算资源,或者排队队列的长度相对较短,则认为此结点为系统中的轻载结点,反之,为重载、或过载结点。通过这几种方式的定义,rwAgent 算法保证了 Agent 可以仅根据局部、本地的信息来决定是"放下"任务,还是携带任务继续移动,而不需要获得网格系统的全局知识。

图 7-21 所示描述了网格结点的多代理组织结构,网格系统中的每个超级结点均由四个部分组成:可供任务使用的网格资源、代理生成器、决定代理采取何种行为的决策器以及负责结点内部资源调度的本地资源调度器。这些网格结点组织在一个具有超级结点拓扑结构的对等网络中,每个结点都是计算对等的。

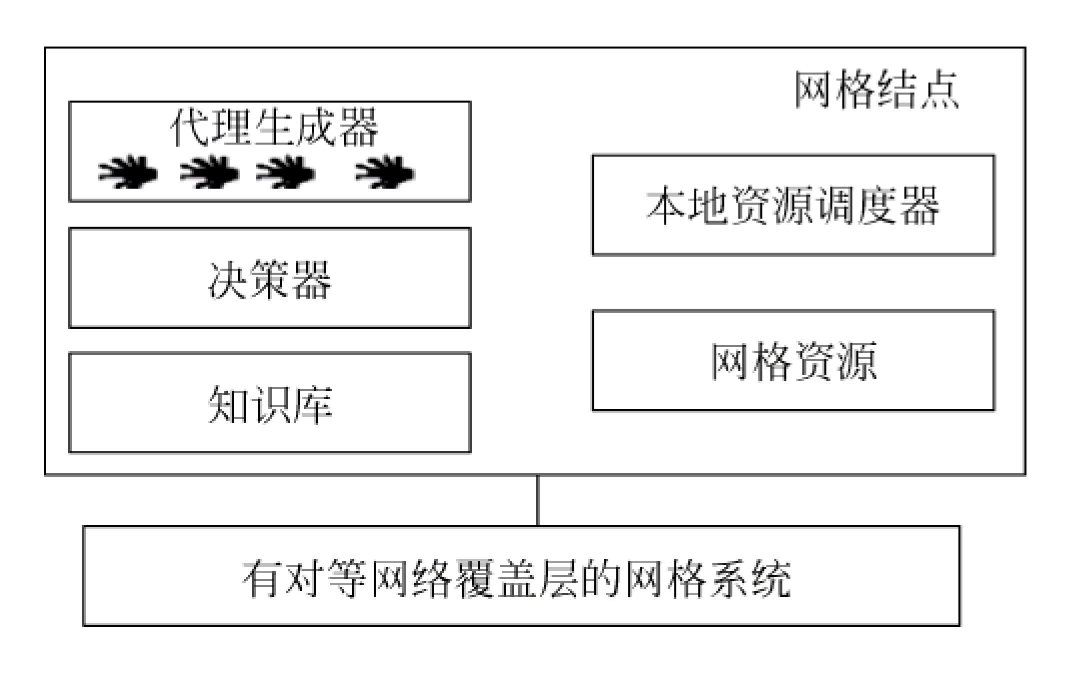


图 7-21　网格结点的多代理组织结构

3. 算法描述

rwAgent 算法的实现基于以下三个假设。

假设 7-3　每个代理都是主动的,可以自主选择是否加入网格结点的排队队列。

假设 7-4　如果选择加入的网格结点的负载较轻,代理会获得一定的正收益。代理不允许加入过载运行的网格结点。

假设 7-5　对单个任务请求而言,系统初始化后,代理的总数量不再改变。

假设某一时刻网格系统中共有 N 个子任务待到调度,网格系统的空闲计算资源使用量化的 CPU 数量和网络带宽来表示。例如,如果某个网格结点当前可提供 m 个单位的计算资源,意味着该结点的 CPU 数量和网络带宽可以用 m 个数量单位来衡量。一个结点上只有一个排队队列 T,T 的最大长度为 n。网络性能使用测定的超级结点间网络延迟的样本平均值。

rwAgent 算法流程图如图 7-22 所示。

下面给出了 rwAgent 算法的伪代码。

```
CS: nodes set that satisfy the requests of the sub - jobs;
|CS|: total number of the idle resource units in CS;
R[i]:array of the idle resource unit amount of node i;
N: number of the sub - jobs;
Node(i): node whose idle resource units is i;
T[i]: array to present the team of node i in CS;
|T[i]|: length of T[i];

Agent - based load balance algorithm {
    CS = Φ;
    M = |CS|;
    I = |T[i]|;
    for each R[i] {
        if (R[i] satisfies the requests of the sub - job) {
            CS ✍Node (R[i]);
            M = M + m;
            }}
    for each T[i] {
        I = I + ⌈N/M * m⌉;
        if I > n{
            One agent leaves T[i];
            I = I - 1;
            for each moving agent{
                j = Chord(i).successor;
                //using Chord routing protocol to find the successor node of i
                if  ( (|T[j]| > n) or (it continues to keep moving) Continue;
                else {
                    T[j] = |T[j]| + 1;
                    Break;
        }}}
        else
            sub - task stays in T[i] and waits to be processed;
}}
```

开始
代理携带子任务在网格中按照Chord路由协议移动
遇到需要计算负载程度的结点
是否重载或过载?
是
否
代理放下子任务子任务加入结点的排队对列等待处理

图 7-22 rwAgent 算法的流程图

rwAgent 可以实现网格系统全局的动态负载均衡，和其他多代理系统不同。例如 ARMS 和 Messor 等，ARMS、Messor 都采用了层次式结构对代理进行管理，而 rwAgent 采用 P2P 的管理方式，代理之间完全对等。同时，ARMS 需要使用 PACE 对结点的性能进行预估计，而 rwAgent 不需要任何的先验知识，是自配置的，代理根据自己在移动过程中获取的知识作为资源调度的判断依据。

7.3.4 数学建模

为了从理论上分析、验证算法设计的正确性和可行性，采用数学建模的方式进行分析，求证是否存在一个网格的全局负载均衡状态，分析为了达到此负载均衡状态而付出的代价是否合理，例如为达到负载均衡状态，代理游走时间、调度结点的等待时间是否过长等。

参考文献[51]使用一个简单的本地策略描述了代理之间的 Coalition Formation 关系，

基于此文的工作，本节设计了一个相似的模型来描述 rwAgent 算法中多代理之间的协作关系。因为 rwAgent 代理的行为是参考文献[51]中代理行为的逆过程。

1．Coalition Formation 模型

在处理一个任务时，使用多代理系统之所以能够获得比使用单个代理更优的性能，其中很重要的一个原因就是多代理之间的协同工作，又称 Coalition Formation。描述代理间的 Coalition Formation，传统的方式是使用协商，但是计算和通信的复杂程度使得这种方式不适应于大规模的系统。参考文献[51]提出了一个本地策略的机制，假设代理是随机移动、相互影响的本地实体。当两个代理相遇时，一个新的 Coalition 形成，如果有第三个代理加入时，此 Coalition 将随代理的个数而增长。使用一个宏观的数学模型来表示代理间的这种行为，模型描述了 Coalition 的个数和分布情况随时间的变化而变化的规律。此模型规定：代理以一定的概率离开 Coalition，因此提高了模型的普遍适应型。

2．rwAgent 模型

在 rwAgent 算法进行网格资源调度时，每个子任务由代理携带着在网格中按照 Chord 路由协议移动，当遇到一个网格计算结点时，代理可以根据当前结点的负载程度以及自己的先验知识来确定是否将此子任务分配给该结点处理。rwAgent 算法的设计思路和参考文献[51]的假设类似，即代理都是主动的，均以一定的概率离开或加入一个 Coalition。不同之处在于：rwAgent 算法中，在适当条件下，代理将任务“放下”，而在参考文献[51]中，代理将“购买”即“拾起”目标实体。除此之外，rwAgent 算法中的代理按照 Chord 协议来决定自己的下一跳路由，而参考文献[51]中的代理在网络中随机游走。可以认为，rwAgent 代理的行为是参考文献[51]中代理行为的逆过程。基于参考文献[51]中的宏观模型，本部分设计了一个类似的数学模型来描述网格资源调度过程中 rwAgent 代理的行为，从而对算法的负载均衡等性能给出理论分析。

在定义数学模型之前，首先给出几个定义。

n 代表某结点任务排队队列的最大长度，在 t 时刻，长度为 i 的队列用 $x_i(t)$ 来表示。

初始化时，系统中共有 N 个代理（子任务），$N \geqslant n$。根据假设 7-3，初始化以后，系统中的代理总数量不变，即 $\sum_{i=1}^{n} i x_i(t) = N$。模型 7-1 描述了随着时间的变化，排队队列长度的变化。

模型 7-1：

$$x'_1 = 2D_2 x_2 + \sum_{k=3}^{n} D_k x_k - 2A_1 x_1^2 - x_1 \sum_{k=2}^{n-1} A_k x_k$$
$$x'_i = -D_i x_i + D_{i+1} x_{i+1} + A_{i-1} x_1 x_{i-1} - A_i x_1 x_i, \quad 2 \leqslant i \leqslant n-1$$
$$x'_n = -D_n x_n + D_{n-1} x_1 x_{n-1}$$

在以上各式中：$\sum_{i=1}^{n} i x_i = n$，$A_j > 0$，$D_j > 0$，$x_j \geqslant 0$，$1 \leqslant j \leqslant n$；$x'_i$ 即 $\mathrm{d}x_i(t)/\mathrm{d}t$，表示 t 时刻队列长度的变化率；加入率 A_i 表示代理加入一个长度为 i 的队列的概率；离开率 D_i 表示代理离开长度为 i 的队列的概率；加入率 A_i 和离开率 D_i 根据代理在网格中执行负载均衡几分钟后的经验值确定。

模型 7-1 能很好地描述网格系统的全局负载均衡状态。例如，第一个等式中的 $2D_2x_2$ 表示一个代理离开长度为 2 的队列后，系统中会形成两个长度为 1 的队列；$-2A_1x_1^2$ 表示当一个代理离开一个长度为 1 的队列，加入另外一个长度为 1 的队列后，系统中会形成一个长度为 2 的队列。当 $3 \leqslant k \leqslant n$ 时，D_kx_k 表示一个代理离开一个长度为 k 的队列，加入一个空闲的队列后，变成一个长度为 1 和一个长度为 $k-1$ 的队列。当 $2 \leqslant k \leqslant n-1$ 时，$-A_1x_1x_k$ 表示当一个代理离开一个长度为 1 的队列，然后加入一个长度为 k 的队列后，系统中形成了一个长度为 $k+1$ 的队列。模型 7-1 的每一项都和负载均衡的特性一一对应。

参考文献[51]已经证明了模型 7-1 存在一个稳定的负载均衡状态，这种状态是 x_1 的函数：

$$x_i = f(x_1) = c_i x_1^i, \quad 1 \leqslant i \leqslant n \tag{7-3}$$

式中：

$$c_1 = 1, c_i = \frac{A_1A_2\cdots A_{i-1}}{D_2D_3\cdots D_i}, \quad 2 \leqslant i \leqslant n$$

从式(7-3)中可以看出，系统的负载均衡程度取决于代理加入和离开结点的概率。

评价一个负载均衡算法的性能，最重要的一个标准是为了达到负载均衡状态而付出的代价是否合理。在评价 rwAgent 算法性能时，主要考虑了算法的总体收益，即为达到负载均衡状态、实现系统的高吞吐量，调度结点的平均等待时间。在每个代理都获得了自身的最大收益时，算法的全局收益最大[52]。

假设 $x^* = (x_1^*, x_2^*, \cdots, x_n^*)$ 是模型 7-1 的唯一解，每一个代理占有同样的服务时间 T，$T>0$。可以证明：在是否需要达到全网格的负载均衡状态的两种情况下，模型 7-1 的总体收益是不同的：

$$\begin{aligned} E &= N(N-1)T/2 - \sum_{i=2}^{n} i(i-1)Tx_i^*/2 \\ &= \frac{1}{2}N^2T - \frac{T}{2}\sum_{i=1}^{n} i^2 x_i^* \end{aligned}$$

根据式(7-3)可以得到：

$$\frac{2}{T}\frac{\mathrm{d}E}{\mathrm{d}N} = \left(\sum_{i=1}^{n} i^2 c_i x_1^{*i}\right)^{-1}\left(\sum_{i=1}^{n} i^2(2N-i)c_i x_1^{*i}\right) > 0 \tag{7-4}$$

由此可知，模型 7-1 的总体收益 $\frac{\mathrm{d}E}{\mathrm{d}N}>0$，随着排队队列最大长度的增大，模型 7-1 的总体收益会越大。

此结论表明：结点的性能越高，可以达到的全网格的负载均衡状态会越好。这对于网格系统的设计和实现非常重要，在组建虚拟组织、选择超级结点时，需要考虑选择那些性能高的结点作为超级结点，例如超级集群等。同样，客户结点的能力也尽可能选择计算能力好的结点。

这个结论是显而易见的，同样的资源管理方式下，完全由高性能计算机组成的网络总是要比仅由几台普通计算能力的计算机组成的网络具有更高的计算能力。

这也从另一个角度进一步证明了使用超级结点对等网络有利于使网格维持一个全局负载均衡状态，从而获得更好的计算性能。

式(7-3)说明，通过调整 A_i 和 D_i 的值，模型 7-1 可以获得一个很好的性能。在 rwAgent 中，代理加入或离开结点排队队列的概率和队列的长度 i、结点的负载能力 $(N/M\times m)$ 是相关的：

$$f\left(\sum n\right)=P\left(k\times\left|i-\frac{N}{M}\times m\right|\right),\quad 1\leqslant i\leqslant n \tag{7-5}$$

式中，k 是调整系数，这个值由代理的经验确定。

7.3.5 算法实验及结果分析

本节我们首先使用仿真的方法将 rwAgent 算法和 Messor 算法进行了比较，包括算法的路由特性、达到负载均衡的时间进行了比较。进一步，为了验证 rwAgent 算法在实际网格环境中的调度性能，在一个网格系统实验平台上，结合具体的计算密集型任务，对 rwAgent 算法的整体性能、任务分配情况、负载均衡度进行了实验测试。

1. 仿真实验

仿真程序使用 GT-ITM 网络拓扑产生器建立了具有超级结点对等网络拓扑的网格环境。GT-ITM 网络拓扑中的 Transit Domain 对应网格中的超级结点，Stub Domian 对应网格中的由超级结点维护的客户结点集合。网格规模为 28800，共有 400 个超级结点。所有结点都通过 SHA-1 安全哈希函数得到，路由表的行数为 8，列数为 8。结点在网络中随机分布。网络连接分为局域网(LAN)和广域网(WAN)，实验中 WAN 的带宽均匀分布在区间[0.5，1.0](单位为 Mb/s)，LAN 的带宽均匀分布在区间[2.5，5](单位为 Mb/s)。仿真程序会周期性地更新客户结点的可用资源信息、索引信息，以此来模拟网格结点的排队队列长度的变。更新数据由 NWS 测定实际应用系统得到，包括结点负载、CPU 空闲率等。设定结点的计算速率按均匀分布随机分布在 200、400、600、800 和 1000(单位为 Mflop/s)之间。这些跟踪数据使得模拟环境能够仿真实际网格环境的资源异构性特征。同时，仿真程序还会周期性地维护超级结点的路由表和邻居信息。实验不考虑通信延迟和通信链路竞争。实验给每个结点赋予一定的生命周期，以此来模拟结点的加入和退出。仿真实验中使用的 GT-ITM 参数如表 7-2 所示。

网格任务的表现形式为对客户结点可利用资源信息的请求。仿真程序假定任务总数为 1000，任务请求的发起过程服从泊松分布，平均到达间隔时间为 8s。任务的输入数据大小相同，输入文件使用实际应用系统的数据，其中，设定任务对资源的需求量按均匀分布随机产生，分布区间为[0.1,100]。

在调度过程中，rwAgent agent 在移动时将 Chord 标识环中当前结点的后继结点作为路由方向；Messor ant 则随机地从路由表中选择一个结点作为移动目标。

1) 平均服务质量 QoS

如 7.1.5 节所述，本实验中使用平均响应率来评价 rwAgent 算法的整体服务质量 QoS。在分析平均响应率时，将 rwAgent 算法和 Messor 算法的平均响应率进行比较。仿真实验中，分别记录同一个任务在不同的调度算法下的执行时间，计算任务的响应率；然后改变任务输入数据的大小，进行了几组测试。图 7-23 记录了任务输入大小分别为 1MB、2MB、3MB、4MB 时，在两种不同的调度算法下，重复执行任务 10 次，获得的实验结果。

从图 7-23 可以看出，两种算法的平均响应率都随着任务输入数据的增加而降低，但是

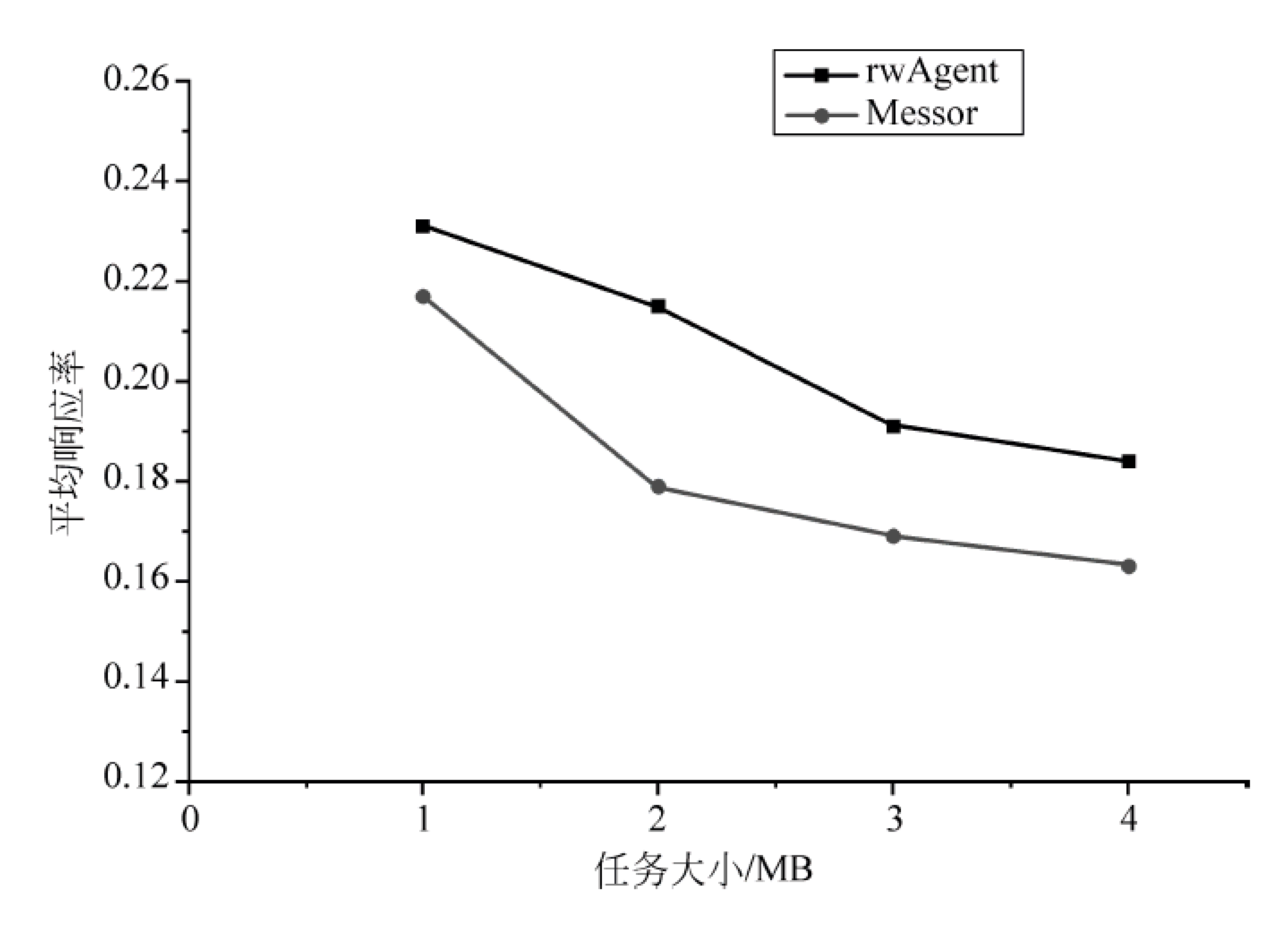


图 7-23 rwAgent 和 Messor 的平均响应率(QoS)比较

rwAgent 算法的 QoS 明显高于 Messor 算法。例如,任务输入数据大小为 1MB 时,rwAgent 算法的平均响应率为 0.231,而 Messor 算法为 0.217。随着输入数据的增大,即任务计算量越来越大的情况下,两种调度算法的平均响应率均有所下降,并且呈现出几乎相同的趋势:随着任务输入数据量持续增大,平均响应率下降速度降低。例如,当任务输入由 2MB 增长为 3MB 时,rwAgent 和 Messor 的平均响应率分别下降了 0.024 和 0.01,而当输入变为 4MB 时,rwAgent 和 Messor 的平均响应率分别下降了 0.007 和 0.006。

对于这种实验结果,可以给出以下两种解释:

(1) rwAgent 和 Messor 两种算法的代理在经过一段时间的工作后,可以根据自己的经验作出正确的判断,从而提高了算法的效率。

(2) 经过代理长时间的协作,两种算法都能够获得较好的负载均衡效果,从而使得模拟系统能够有较高的吞吐量,提高了调度性能。

从实验结果可以看出,rwAgent 算法的 QoS 明显高于 Messor 算法,分析其原因,可以认为:Messor 算法为了追求好的负载均衡效果,需要 ant 在 SearchMax 和 SearchMin 两个状态间进行切换,虽然整个系统的负载可以达到良好的均衡状态,但是无疑增加了开销。而 rwAgent 算法虽然也力求保持系统的全局负载均衡,但是首先要确保用户可获得的 QoS,例如,对用户而言,任务运行时间越短越好。因此,rwAgent 算法在全局负载均衡状态和任务执行时间之间做了折中,规定 rwAgent agent 不会处理已经调度过的任务,即 agent 不会为了均衡结点间的负载而对以调度的任务进行二次调度;代理仅根据自己的先前经验知识来确保新任务调度的负载均衡效果。这点与 Messor 算法代理需要为了均衡结点间的负载而进行任务迁移是完全不同的。实验结果证明:rwAgent 算法的折中设计可以保证任务高到达率时,算法仍能获得较高的响应率,这一点对于网格终端用户而言至关重要。由以上分析可以得出结论:rwAgent 算法的平均响应率优于 Messor 算法。

2) 路由效率

rwAgent agent 移动时遵循 Chord 路由协议,而 Messor ant 则采取游走(wonder)的方式,因此,设计了这个实验来比较两种调度算法的路由效率。仿真实验中,记录 rwAgent 算

法在调度任务时 agent 所需的平均路由跳数，然后和 Messor ant 执行负载均衡时的平均路由跳数进行比较，如果 rwAgent 算法的路由跳数小于 Messor 算法，则认为 rwAgent 算法的路由效率高于 Messor 算法。通过这种方法来测试 rwAgent 算法 Chord 路由协议进行任务调度的设计是否有效。测得的实验结果如图 7-24 所示。

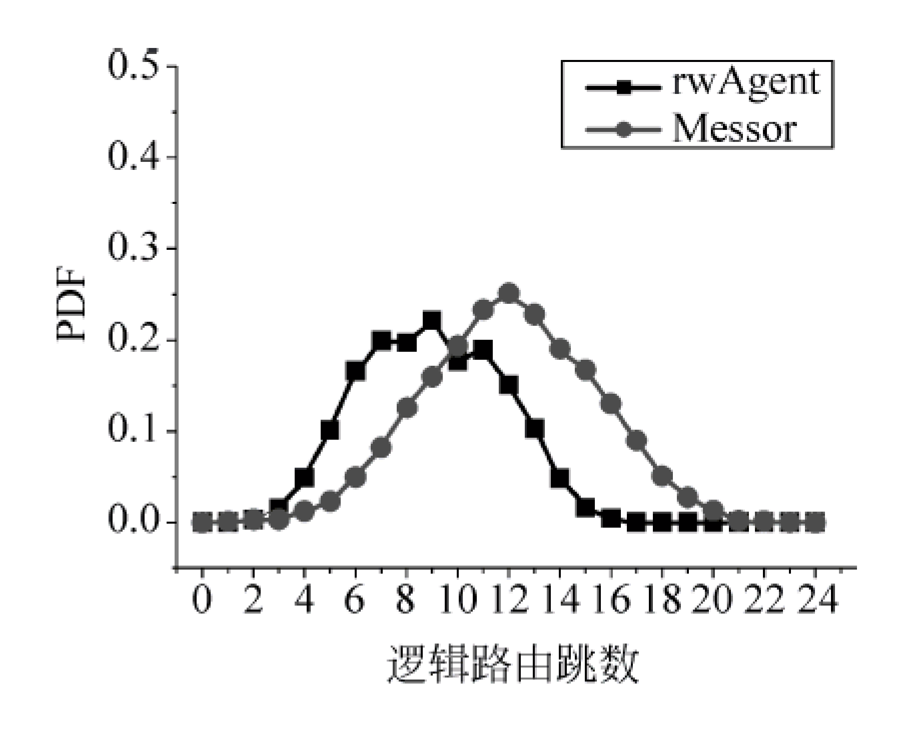


图 7-24　rwAgent 和 Messor 的路由跳数概率密度函数

图 7-24 将 rwAgent 算法和 Messor 算法的路由跳数的概率密度函数进行了比较。rwAgent 算法 PDF 最大时的路由跳数为 9，对应的 PDF 的值为 0.22；Messor 算法 PDF 最大时的路由跳数为 12，对应 PDF 值为 0.25。图 7-30 的实验数据反映出，rwAgent 算法的路由跳数明显小于 Messor 的路由跳数。说明和 Messor 算法的随机游走设计相比，rwAgent 算法采用 Chord 路由协议来决定 agent 移动方向的设计是正确有效的。

2. 网格实验系统测试

网格实验系统共包括五个网格结点：一台 SGI Onyx3800 集群和四台 PC 机，各结点通过上海交通大学的 100Mb/s 校园网连接。SGI Onyx3800 集群位于上海超级计算中心上海交大分中心。四台 PC 机的配置为：1GHz Pentium 4 处理器，256MB RAM，40GB 硬盘。

在实验过程中，向网格系统提交一个大的蛋白质分子，进行 DOCK 计算，以此来作为网格任务。样本分子数据库采用 Specs 数据库。

在进行 DOCK 实验过程中，使用 rwAgent 算法对实验系统中可利用的网格计算资源进行调度，实现全网格的负载均衡。网格结点的负载程度使用空闲 CPU 数目和可提供 CPU 数目的比值来表示。空闲 CPU 数目使用实时测得的系统 benchmark 值表示，量化为当前可利用计算单元的个数，此 benchmark 值在实验系统初始化时确定，并且保持固定不变。

1）执行时间 makespan

使用任务总体执行时间 makespan 来评价 rwAgent 算法的整体调度性能。为了分析算法的性能，将实验结果和不使用任何调度策略、使用调度策略但不考虑负载均衡，两种情况下测得的任务总执行时间进行了比较。具体做法是：对于同一个 DOCK 实验，首先在一台集群上进行计算，此过程不使用任何网格调度策略，只是单纯地将任务以并行的方式处理，记录从任务提交到任务完成所需计算的时间。然后，再将同一个任务提交到网格实验系统中，使用 rwAgent 算法进行调度，主要以实现全网格的负载均衡为目的，记录任务的总执行

时间。最后，在分析计算结果同样有效的情况下，比较两种不同的调度模式下任务所需的计算时间。改变任务的输入，提交不同的大分子，或者选择 Specs 数据库中不同的样本分子，重复上述过程，进行了多组实验。图 7-25 记录了将同一个大分子和 Specs 数据库中的 1000、5000、10 000 个样本分子进行 DOCK 匹配计算，分别在 Sunway32A 上进行并行计算和在网格实验系统上使用 rwAgent 调度算法的情况下，测得的任务执行时间。每种情况的实验数据是同样条件下重复 5 次的平均结果。

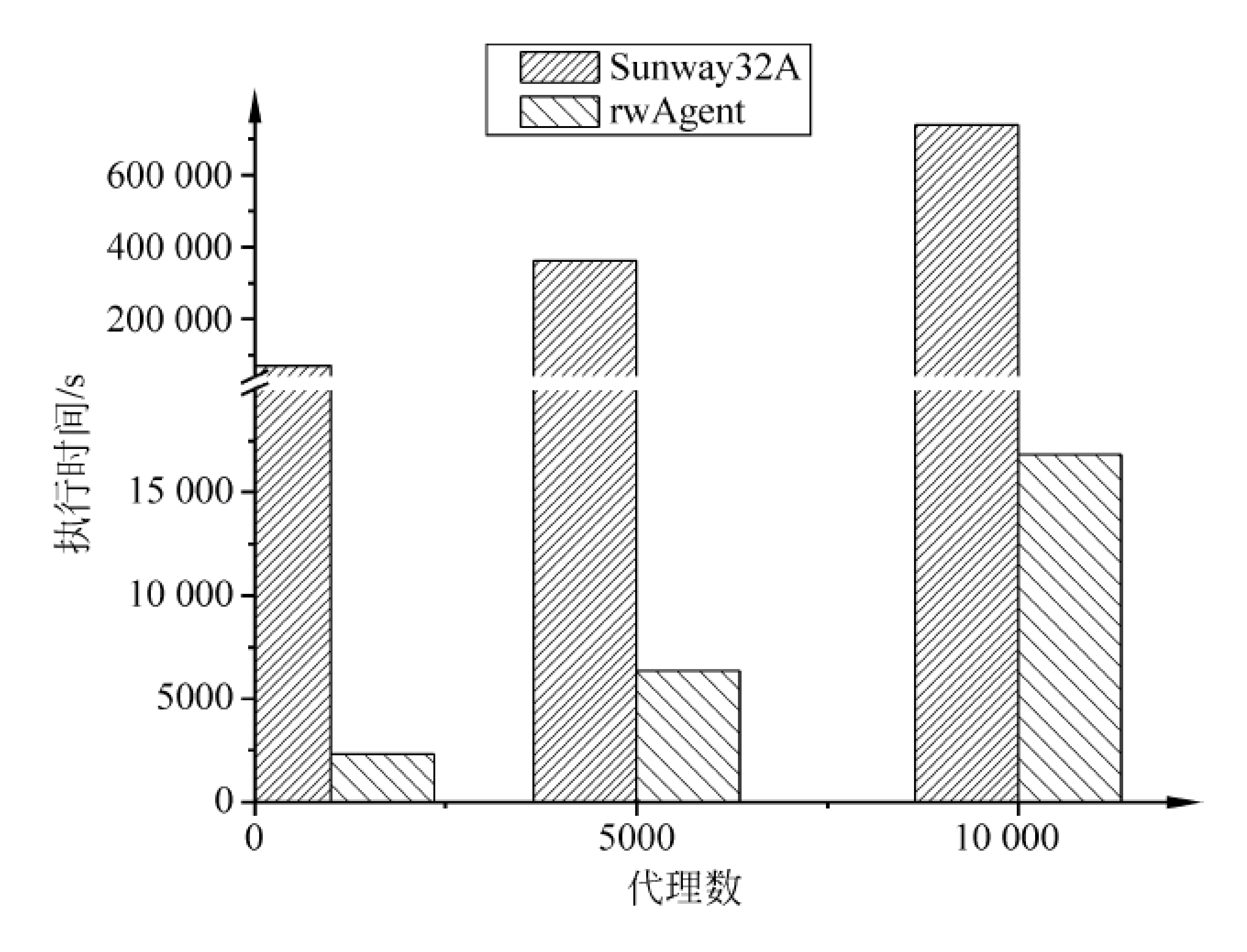


图 7-25　不同调度模式下任务的执行时间

从图 7-25 可以看出，得到同样有效的计算结果的条件下，选择 1000 个样本分子进行 DOCK 实验，在不使用调度策略的情况下，需要在 Sunway32A 上运行 71 398.12s，在网格实验系统上使用 rwAgent 负载均衡算法，需要 2311.56s 的计算时间。随着选择的 Specs 数据库中样本分子的数量增多，例如，样本分子数量为 10 000 时，需要在 Sunway32A 上计算 737 591.12s，网格实验系统中，rwAgent 负载均衡算法仅需要 16 835.92s。

计算两种不同调度方式的加速比，得到 rwAgent 对 SGI Onyx3800 的加速比分别为 30.88、57.09、43.81。由此看出，随着样本分子数量的增多，计算时间变长，经过代理自身经验的积累，rwAgent 算法呈现出更好的性能。

由此实验分析可以得出结论：rwAgent 算法可以进行有效的网格资源调度，明显加快了计算的进程；通过多代理之间的协作，提高了系统的吞吐量。

2）负载均衡度（Load Balance Level）

为确保网格系统的负载均衡，rwAgent 算法必须避免把任务分配到工作负载重的计算结点上，同时，为了保持系统的可用性，也不能把结点的计算资源一次耗尽。因此，我们希望和 4 台 PC 机相比，吞吐量、计算能力都相对高的 SGI 机群可以承担大部分的任务。因此，本小节设计了测量系统负载均衡程度的实验。实验方法如下：向网格实验系统提交一个蛋白质大分子，选取 Specs 数据库中一定数量的样本分子进行 DOCK 计算；在指定的观测时刻，记录每个计算结点用于进行 DOCK 实验的计算资源数量。因为每个子任务的大小是相同的，因此，结点参与计算的计算资源数量可以反映出 rwAgent 算法分配给结点的计算任务量，从而可以评价系统的负载均衡程度。

为了验证式(7-5)的正确性，即代理加入还是离开队列的概率会影响网格系统能达到的负载均衡程度，我们记录了 A_i，D_i 和调整系数 k 的取值情况，如图 7-26 所示。例如，在 5：00am 的观测时刻，rwAgent 算法的各个参数取值分别为 $N=3000$，$A_1=0.0001$，$A_i=D_i=9$，$1<i<3000$，$k=0.021$；而在 13：00am 时刻，$N=3000$，$A_1=0.0001$，$A_i=D_i=6$，$1<i<3000$，$k=0.045$。通过比较这两个不同时刻的各结点的状态，发现两种情况下 DDGrid 网格的负载均衡程度是不同的。因此，可以做出结论，实验结果验证了式(7-5)的正确性，基于多代理协同计算的 rwAgent 算法可以使网格系统获得很好的负载均衡特性。

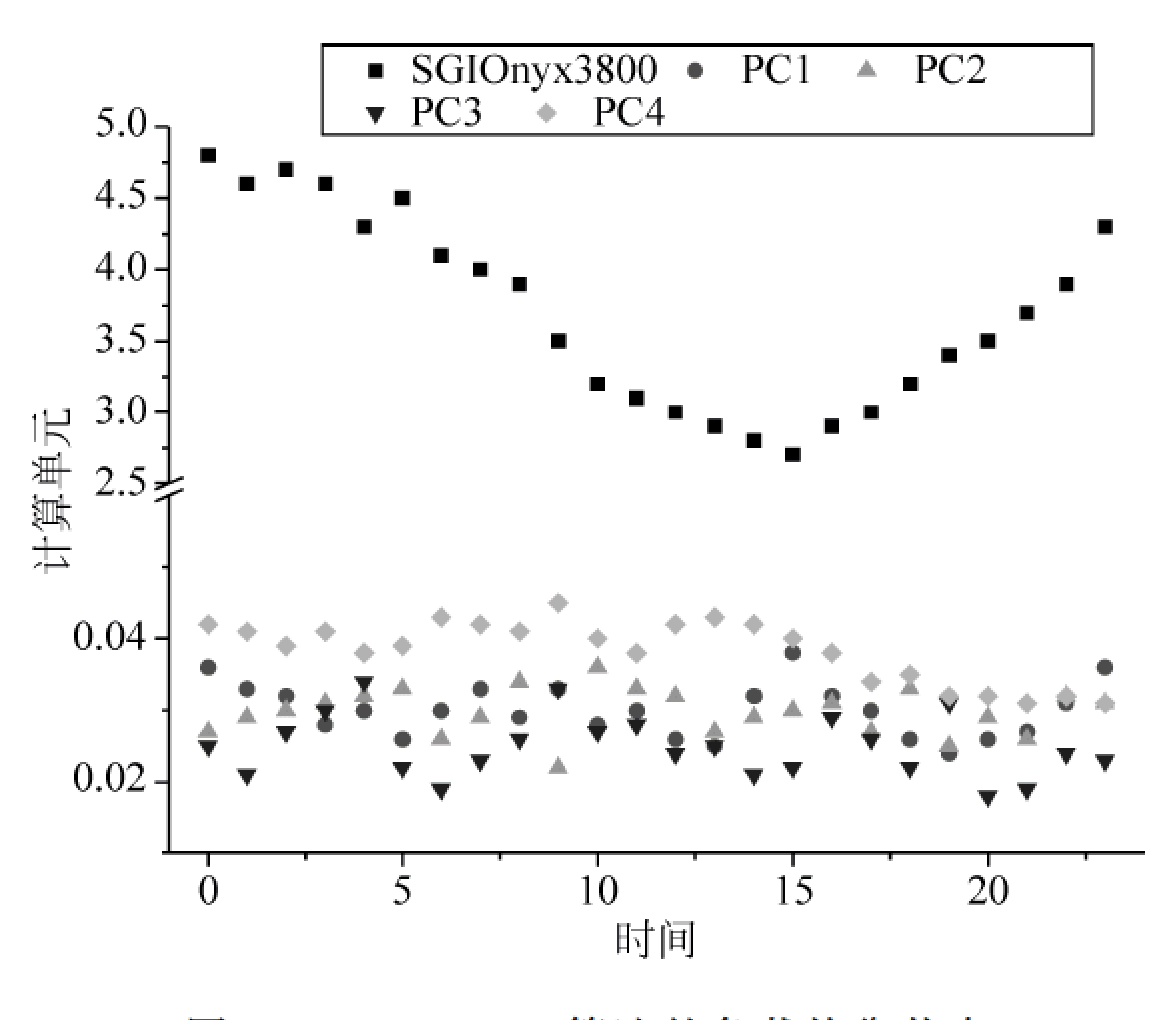


图 7-26　rwAgent 算法的负载均衡状态

7.4　资源管理和调度算法在新药研发网格中的应用

本章中所有实际运行环境中的实验数据都是在新药研发网格平台 DDGrid 上测得的。结合本书的研究工作，在开发过程中，Internet Computing 实验室网格与 P2P 研究小组的其他成员、江南计算技术研究所高性能计算组、上海药物所、香港大学 ETI 研究所以及香港大学计算机系的研究人员提出了许多宝贵的意见，并给予了很多无私的帮助①。DDGrid 是一个利用分布在 Internet 上的高性能工作站的过剩计算资源进行药物筛选计算的计算网格系统，它提供的功能包括网格系统资源的管理、调度、任务监控、计费统计和用户管理等。它的主要组件是用 C++ 语言实现的，具有跨平台、高可扩展等特性。本节首先介绍了 DDGrid 的体系结构，然后对它的一些核心组件进行剖析，最后通过一个实例阐明了它的应用前景。

7.4.1　新药研发网格项目背景

药物研究和医药产业在国民经济发展过程中起着举足轻重的作用，可以带来巨大的经济效益和社会效益，医药产业的利润率很高，高居全球企业的榜首，第二位和第三位的分别是电

① DDGrid 的含义是新药研发网格(Drug Discovery Grid)。DDGrid 的研发是国家 863 网格重大专项“高性能计算机及其核心软件”(No. 2002AA104270)的重要子课题。

讯业和计算机产业。同时,药物研究可以实现人口与健康领域的国际目标。无疑,医药产业将是我国21世纪新的经济增长点和支柱产业。然而,新药研发和发展的过程周期长、耗资巨大,一种药物从随机筛选化合物开始,到最后上市,需要10～12年的时间、3.5～5.5亿美元的经费。图7-27是新药研发和发展的基本流程。

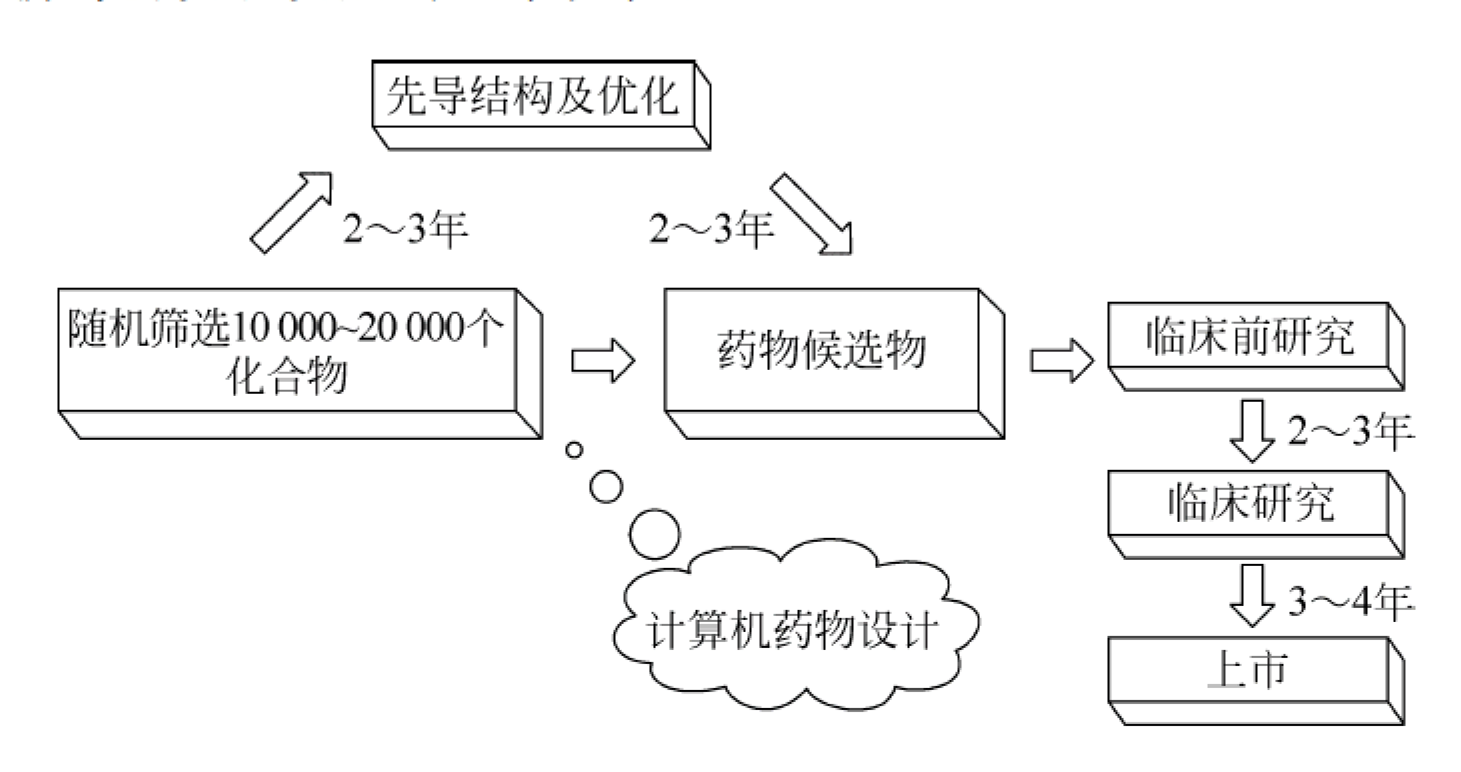


图7-27　新药研发和发展的流程

从图7-27中可以看出,新药研发的核心之一是从大量的化合物样品库中发现有药理活性的化合物,此过程称为药物筛选。进行药物筛选的方式分为两种:常规药物筛选和虚拟药物筛选。虚拟药物筛选又称计算机药物筛选,需要首先通过计算机筛选,使得实际筛选的化合物集中几千倍,也就是说通过计算机对几万个化合物数据库的筛选,合成上百个化合物进行实物筛选即可。国际上公布的计算机药物筛选的命中率为2%～24%,而常规药物筛选的命中率仅为0.01%～0.001%。所以计算机的应用不仅可以成千上万倍地提高药物筛选的有效性,同时可以大量减少新药研制的成本和时间。虚拟药物筛选需要大量的计算资源。

利用高性能计算机进行虚拟药物筛选是现阶段普遍采用的一条途径。

使用高性能计算机研究发现活性化合物结构,并对上述化合物结构重新组合生成新化合物数据库,重复上述虚拟药物筛选,从而发现新的性能更好的化合物。这种虚拟药物筛选的方式可以提高药物筛选的有效性,缩短新药研发的成本和周期。但是,使用计算机进行药物筛选也面临如下一些问题:

(1) 化合物分子数据库的规模增大,今后几年将增加到1000万个化合物左右,计算量会相应提高5倍。

(2) 由于计算机组合化合物库设计与计算机药物筛选相结合,计算机将设计大量的虚拟化合物供药物筛选,进一步提高计算量。

(3) 人类基因组计划的完成,将提供大量的可用于疾病治疗的分子靶标,使得计算机药物筛选的工作量提高几十倍。

(4) 目前计算机筛选的正确性仍然处在定性水平,随着方法的改进,将向半定量和定量化发展,计算量将大幅度增加。

虽然高性能计算可以大大提高药物筛选的效率,然而,随着越来越多的应用提交到集群上,集群的工作负载越来越重时,终端用户等待计算结果返回的时间会越来越长。这个问题最直接的解决方案是改善硬件的配置,但是这种方案费用高,并且不能够从根本上解决问

题。因此，迫切需要新一代的计算技术支撑新药研发中的虚拟药物筛选。

研究表明，Internet 上有很多闲散的计算资源，并且很多用户愿意将自己的空闲资源贡献出来[7,53]，因此可以充分利用现有的分散在 Internet 上的集群的空闲资源来处理新药研发领域的应用，而不需要购置新的硬件。同时，网格计算提供了资源共享、资源虚拟化的技术，终端用户可以如同访问一台设备一样，直接使用网格中各种异构资源而不必关心它们所在的位置、底层结构。现阶段已经有很多网格系统在处理科研领域的数据密集型应用[54]，例如，SETI@home、Condor 和 NetSolve。这些技术使得采用一种全新的途径来解决上述问题成为可能。因此，我们设计并实现了新药研发网格 DDGrid，旨在利用分布在 Internet 上的集群贡献的空闲计算资源，利用网格技术，结合所做的资源管理和资源调度方面的研究成果，来处理药物筛选领域的数据密集型应用。

7.4.2　相关工作

受到 SETI@Home“寻找外星文明”计划的启发，已经开发出一些@Home 的分布式药物筛选软件，如 Oxford 大学的 Life-saver@Home、Scripps 研究所的 FightAIDS@Home 等。这些软件都是基于 Internet 的 P2P 对等计算，利用 Internet 上的 PC 机空闲资源进行药物筛选的研究。每一个参与者下载一个程序，在 PC 机空闲时该程序作为屏幕保护程序进行运算，计算结束后将计算结果发回服务器。例如其中的一个癌症药物筛选计划，吸引了全球逾 150 万台 PC 机参与计算，共筛选了 35 亿的化合物分子。然而@Home 的 P2P 计算仍然存在其不足之处，首先，药物筛选的计算周期不能保证，另外，@Home 不能提供服务，这种缺陷限制了它的进一步应用。

7.4.3　DDGrid 的体系结构

DDGrid 利用网格环境和网格计算资源进行药物筛选，并且提供筛选服务。平台具有统一应用网格界面，用户经身份论证后，只需提交作业即可获得结果而无须考虑如何实现。该网格最大限度地利用中国国家网格的空闲计算资源，无论是哪个计算中心的并行计算机，只要加入 DDGrid，其空闲计算资源就可能被 DDGrid 利用。如果某一时刻其计算资源繁忙，药物筛选计算就会自动暂停，让出计算资源，待到有足够的计算资源空闲时重新启动计算。因此，DDGrid 需要通过网格结点内和结点间的进程调整和负载控制来完成计算资源的调度，最终完成任务的计算，返回计算结果。这是一个复杂的过程，需要详细的规划。首先应从需要网格提供的服务开始，仔细考虑网格的基本结构，如何使这些基础架构模块有效地融合在一起，这些基本的模块包括安全、资源管理、信息服务和数据管理，这些将影响到应用的架构设计和配置。

图 7-28 所示是 DDGrid 的体系结构。从图中可以看出，这是一种分层式、可扩展的系统结构。逻辑上可以将整个系统划分为三层：应用层、网格中间件、网格结点。从功能上看，系统主要提供任务提交、任务调度、任务计算等主要功能。

在应用层上，分布式网格门户实现了网格任务的自动提交功能，其分布式设计加强了系统的鲁棒性，从而使系统可以同时接受多个网格任务的提交，并且采用了几种方法来优化网格通信，例如限制合法用户上传文件的大小、采取数据加密压缩的方式进行结果回传等。

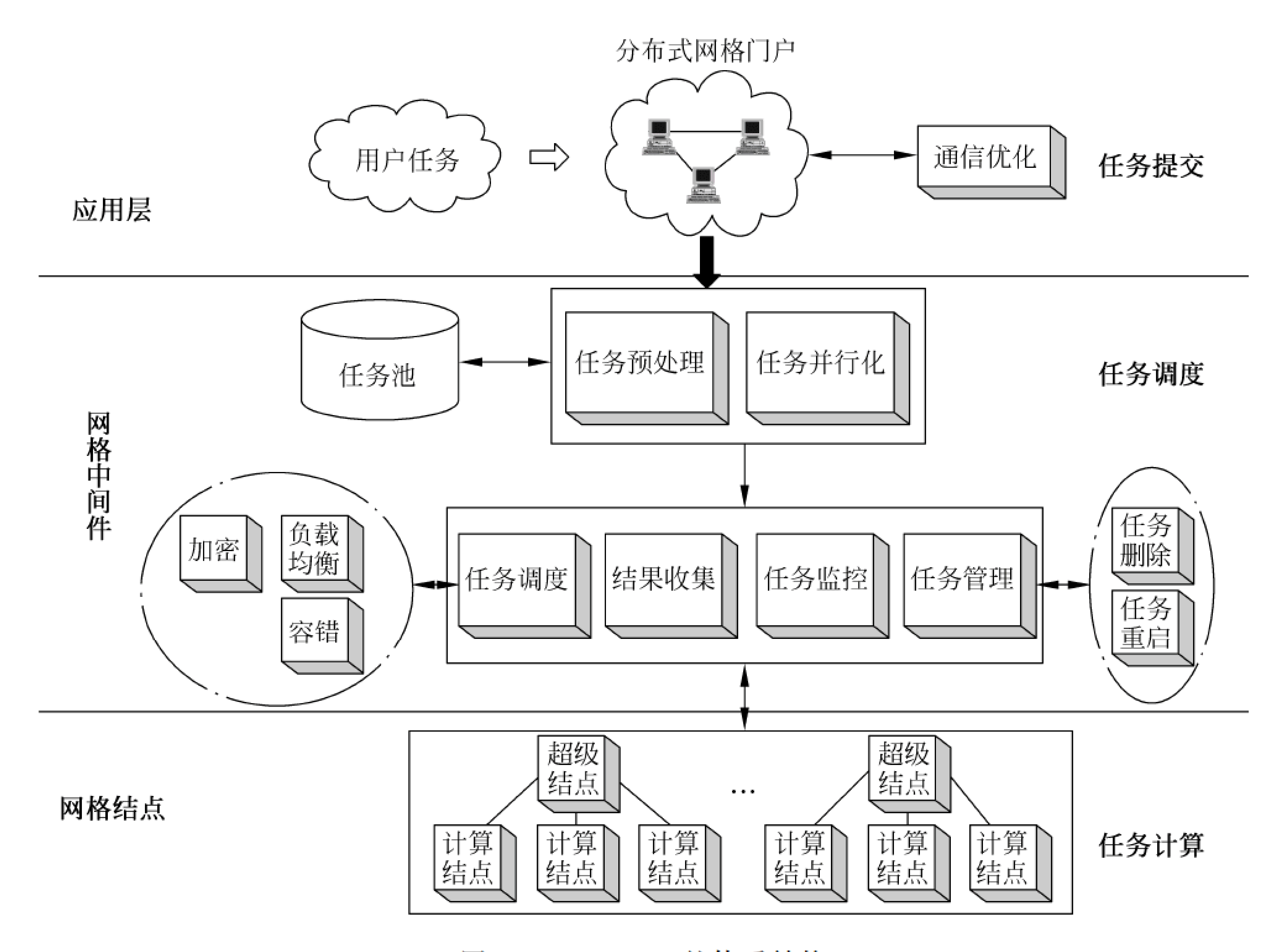


图 7-28　DDGrid 的体系结构

任务调度层首先对提交到网格中的任务进行数据提取、预处理和并行化，形成多个并行的子任务；然后通过一定的调度策略对这些子任务进行调度，调度过程中能够保证数据的安全、实现网格系统的负载均衡；计算结束后可以进行结果收集，形成有效的计算结果返回给用户；任务管理模块允许用户对自己的任务进行删除、同时允许系统管理员对任务进行重新启动；系统还能够对任务的执行情况进行监控，包括任务的进度、资源分配情况、所使用的计算资源数量等，这些数据都将是网格计费管理的重要依据。

调度结束后，处在任务计算层中的网格结点开始计算分配来的子任务，计算结束后将结果回传给任务调度层。在此层中，网格结点组织在一个超级结点对等网络中，结点分为超级结点和其所辖域内的计算结点。通过超级结点可以对分配到所辖域内的子任务进行二次调度，因此可以实现 DDGrid 系统的二级负载均衡。当前，DDGrid 能够最大限度地利用中国国家网格的空闲计算资源来为蛋白质分子对接 DOCK 实验提供计算服务。将来，DDGrid 将能够利用 Internet 上各种可利用资源来为虚拟药物筛选领域的多种数据密集型、计算密集型应用提供服务。

7.4.4　主要组件设计

由于整个系统的设计与实现涉及很多内容，对这些组件进行逐个剖析已超出了本书的范围，因此下面只对其中的一些关键组件进行分析，包括分布式网格门户设计以及进行网格任务调度过程中涉及的一些关键技术。

1. Grid Portal 设计

DDGrid 的 Grid Portal 采用了分布式的系统结构，从而可以同时处理多个用户提交的网格任务，并且加强了系统的鲁棒性，防止因为一个结点失效造成整个网格不能访问的情况发生，提高了网格的访问能力。

如图 7-29 所示，Grid Portal 可以提供的主要功能包括用户管理、任务管理、任务监控、结果下载等。

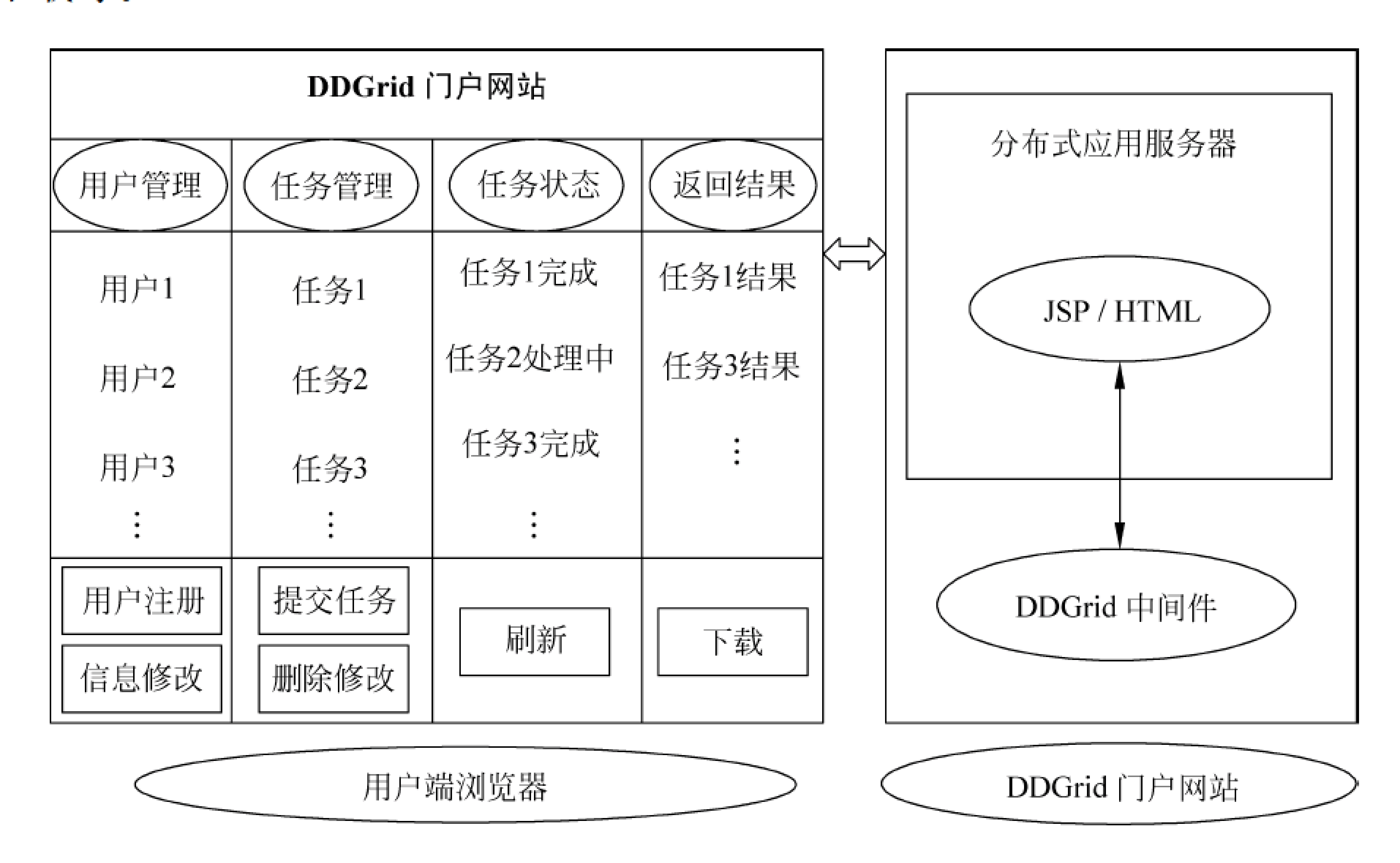


图 7-29　分布式网格门户的功能模块

DDGrid 的用户管理模块具有新用户注册、用户信息修改等功能。只有合法的注册用户才能够使用 DDGrid 提供的药物筛选服务，即提交网格计算任务、下载计算结果、查看网格建设情况等。合法用户还将获得一个自己的密钥，该密钥由网格安全管理模块根据用户信息自动生成，用户使用该密钥才可以得到计算结果的明码。

注册用户可以提交计算任务到网格，在任务提交后，用户具有对此任务进行删除、重新计算的操作权利。如果用户将此任务删除，网格将回收此任务目前占用的资源，同时将部分已经计算完成的结果丢弃。

在任务的计算过程中，用户还可以对任务的计算进度进行监控，例如查看该任务的调度情况、目前的进度、预计的完成时间等。

在计算结束、结果收集完成后，DDGrid 会将有效的任务计算结果返回给 Grid Portal，用户可以从自己的浏览器上看到任务对应的有效计算结果已经生成，并且可以从网格下载。

在系统的设计和实现中，我们充分考虑了数据安全性问题。因此，在结果传递过程中，我们会首先将结果文件加密，然后提供加密后的结果给用户下载，这样即使有人恶意地对结果文件进行拦截，也会因为没有密钥而很难得到明文表示，增强了系统的安全性。用户将计算结果下载到本地后，可以使用自己的密钥将结果文件解密，得到计算结果的正确表示形式。

在网格计算中，网格结点之间的通信量是影响网格系统性能的一个很重要的因素。如果网格结点间存在频繁、大规模的数据传输，这将占用大量的带宽，从而大大降低网格系统的整体性能。因此，DDGrid 设计过程中采用了几种方法来优化网格通信。首先，限制合法用户上传文件的大小，规定用户上传的任务文件的大小不能大于 1MB。其次，采取数据加密压缩的方式进行结果回传，用户下载完计算结果后，必须首先进行解压缩和解密才能使用。再次，要求 DDGrid 上运行的计算任务具有高并行性、各子任务之间通信量尽量少的特点。因此在选择应用实例时，选取了蛋白质分子对接 DOCK 实验，因为 DOCK 实验高度并行，各子任务之间不存在相互通信，非常适合于在网格环境运行。

2. 网格任务调度

如何在网格环境中进行高效的任务调度，使得任务可以利用最少的资源、能够在最短的时间内完成计算，并且这一过程对用户完全透明，同时还允许应用层的用户可以删除自己已提交的任务，这些都是我们在设计网格任务调度方法时需要考虑的因素。

在处理用户任务时，充分利用了本书前几章在网格资源管理和资源调度方面的研究，例如基于资源服务匹配的 GChord 算法、基于多代理协同计算的 rwAgent 算法等。下面将对 DDGrid 任务调度策略进行详细介绍。

1）并行任务划分方法

如前所述，药物筛选领域的蛋白质分子对接实验，又称 DOCK 实验，目的是识别药物靶点。将一个蛋白质大分子和数据库中的样本分子进行结构匹配，计算出样本分子和大分子的某一空间位置上的匹配度，如果匹配度较高，则该位置就有可能是一个药物靶点。一般在进行 DOCK 实验时，用户往往上传一个大的蛋白质分子（病毒分子），选择样本数据库中一定数量的样本分子，例如 100 个样本分子，然后将病毒分子和样本分子进行结构匹配，以此来发现具有药理活性的分子和药物靶点。DOCK 计算结束后，用户将得到一个结果列表，此列表将病毒分子和 100 个样本分子的匹配程度按照从高到低的顺序排列。排在最前面的样本分子最具有成为杀死此种病毒的药物的可能性，而对应的匹配位置就是一个药物靶点。

根据 DOCK 计算的特征，在设计任务调度策略之前，先设计了将任务划分成多个子任务的并行任务分割方法，建立了子任务池。图 7-30 为 DDGrid 子任务池的示意图。

A_Specs_51_ 2005030 8102132	A_Specs_52_ 2005030 8102132	…	A_Specs_100_ 2005030 8102132	A_ACD_201_ 2005030 8202132	A_ACD_202_ 2005030 8202132	…	A_ACD_250_ 2005030 8202132	…	…

图 7-30 子任务池

在建立子任务池的过程中，以样本分子的个数作为划分依据，也就是说，对于一个 DOCK 任务，用户选择了多少个样本分子，那么该任务就将被划分成多少个子任务。例如，用户 A 选择将大的病毒分子分别和 Specs 数据库中标号在[51,100]区间内的 50 个样本分子、ACD 数据库中标号在[201,250]区间内的 50 个样本分子进行对接，那么，该用户上传的任务将被网格系统划分成 100 个子任务。这 100 个子任务分别被赋予不同的标记后，放置

在子任务池中，等待调度算法分配相应的网格资源，进行计算。在对子任务进行标记时，使用“<用户名_样本数据库名_样本数据库中的分子标号_任务上传时间>”作为子任务在任务池中的唯一标记。

2）任务调度策略

相对于大的病毒分子而言，样本分子很小，用来表示样本分子的三维空间结构的矩阵大小基本相同，因此在进行 DOCK 计算过程中，每个样本分子所需的计算量基本相同。基于此，我们设定每个样本分子需要相同的网格计算资源。虽然这会存在一定的误差，但是该误差对于整个网格系统的调度性能影响甚微，可以忽略不计。那么，从用来描述任务池的任务信息表中，以获得各个子任务的名称、计算内容等基本信息之外，还能够估算出计算此任务所需的计算资源数量。

此外，建立了超级结点信息表和客户结点信息表，分别用来描述各个网格结点的详细信息。例如，结点所在的主机名、操作系统版本、当前可用的空闲计算资源数量等信息。

DDGrid 的调度策略采用了和 rwAgent 算法相似的设计思想，资源的调度由多个代理完成。每个网格计算结点都具有自己的代理，该代理知晓结点的当前状态信息。代理定期查看任务池的状态，如果任务池中有等待处理的子任务，则代理会将自身的可用资源和子任务的计算量进行比较，如果自身的空闲资源可以满足一定数量的子任务的计算要求，则代理将对应的子任务取走，并将其加入到对应的计算结点的排队对列中等待计算。子任务在计算结点上进行计算的过程中，如果计算结点的资源繁忙，该子任务将暂停，让出计算资源，等到有足够的空闲资源时再重新启动该进程。

在实现的过程中，设计了多个数据库表格，利用对数据库的访问，来实现代理对资源信息的维护、资源调度过程中的资源匹配、任务的计算状态维护、计算结果收集等一系列相关功能。图 7-31 所示给出了资源管理和任务调度相关部分的数据库设计。

在进行任务调度时，代理会定期查看子任务池中是否存在等待调度的子任务，对应子任务状态表中的“处理状态”项，如果存在值为 0 的子任务，则表示有子任务等待调度。代理从超级结点信息表中提取“可利用资源”表项的值，判断该值是否大于计算单位子任务所需的计算资源数。在大于的情况下，从子任务池中第一个尚未被分配的子任务开始，提取 n 个子任务，分配给此超级结点，其中

$$n=\lceil \text{value(可利用资源)/单位子任务所需计算资源数} \rceil$$

相应地，修改子任务状态表中的“处理状态”表项的值为“1”。

在子任务分配给超级结点后，客户结点的代理会以同样的工作原理进行子任务二次分配，即定期查询超级结点中是否存在等待调度的子任务，然后根据客户结点当前可利用资源的情况判断可以承担的子任务计算量。

超级结点信息表和客户结点信息表中的“可利用资源”项的值会定期更新，以此来达到保证反映结点的实时工作负载情况、实现网格系统的动态资源调度。频繁地更新会浪费网络带宽和系统资源，影响计算速度。但是过慢的更新又将引起资源信息的过时。所以，在 DDGrid 的设计中，选择使用定时更新和事件触发两种方式进行信息表的更新。例如，在上传计算结果、查询子任务池等系统事情发生时，将触发对信息表的维护，以此来保证网格信

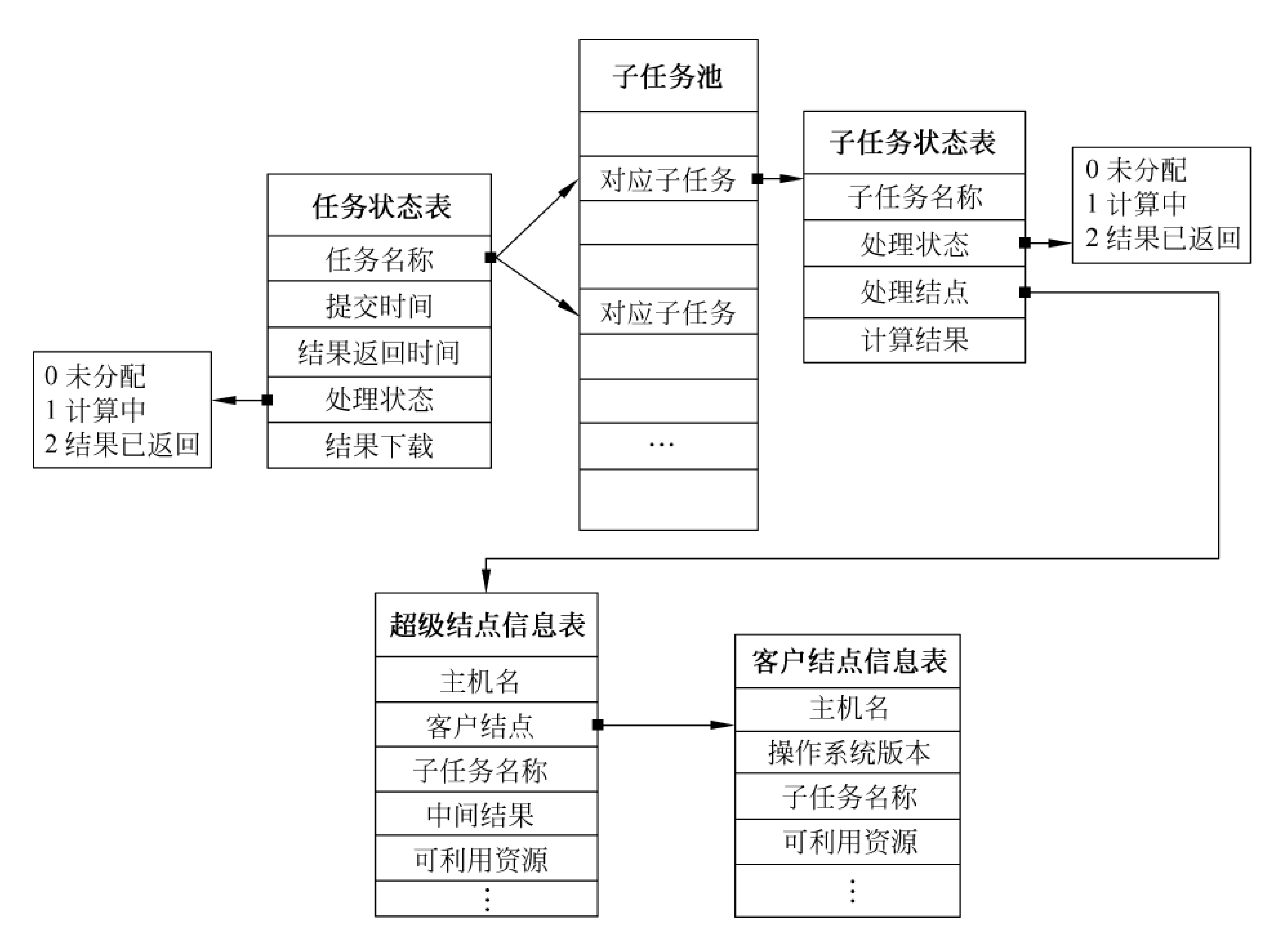


图 7-31　数据库设计示意图

息的实时性。实现过程中，采用 push 和 pull 两种方式来更新信息表。

网格系统的广域性，使得应用程序的执行情况实现难以预测。为了保证整个任务的计算周期，防止因为某个网格结点的崩溃造成结果丢失、任务计算时间过长等情况的发生，在设计过程中，我们采用了基于对等计算的任务复制策略，确保用户可以在短时间内获得正确的计算结果。具体设计思路为，如果某个任务的计算结果在一定时间内没有返回，系统会将该任务的状态重新置为"0"，也就是把此任务作为新的任务进行重新调度。那么，在一定的时间内，系统中可能存在针对该任务的多个调度结果，即由调度器将同一个任务映射到不同的资源域进行计算，那么，在有效时间内，把首先返回的计算结果作为此任务的有效计算结果，而将多个后续返回的计算结果丢弃。

7.4.5　应用实例

图 7-32 是 DDGrid 的运行界面。目前，用户可以访问 DDGrid 的 Web 站点进行蛋白质分子 DOCK 实验，我们仍然努力对它不断进行完善、提供更好更多的服务，希望在不久的将来它能成为医药研发领域用户的一个常用的计算机虚拟筛选工具。

实验使用新药研发网格进行蛋白质分子对接实验，其中使用了 Specs、ACD、CNPP、NCBI 和 ACD-SC 等商业分子数据库，现阶段这些数据库由新药研发网格免费提供给用户使用。



图 7-32　DDGrid 的运行界面快照

7.5　本章小结

7.1 节针对为解决任务并行度高的计算密集型科学应用而设计的专用计算网格提出了基于树匹配的网格资源调度算法 nTreeMatch。算法充分利用了网格资源信息有向图表示方法和网格任务 DAG 表示方法之间的相似性，将网格 Overlay Network 拓扑的最小生成树和 n 叉任务树进行匹配，以实现网格资源的实时调度。提供者的实时资源属性，进行动态的网格任务调度，可以获得较好的负载均衡效果。在构造 Overlay 时借助了网络临近原则同时算法充分利用了 Overlay 网络拓扑提供的路由信息，使用网络邻近原则使得 Chord 环上的邻居结点同时为物理网络上邻近的结点，从而以轻量附加开销来有效减少 Overlay 层上的路由跳数，使得 Overlay 层上的路由跳数尽量接近 IP 层上的路由跳数，降低 RDP，并最终达到提高算法效率的目的。大量的对比实验结果表明，算法可以有效地对网格并行任务进行动态调度，提供较高的服务质量。

7.2 节介绍了基于结构化 P2P 的网格资源管理和调度算法 GChord，该算法的特点是：具有自组织、可扩展和自适应等优点；通过资源发现的方式进行网格任务的动态调度，解决了采用信息收集方式进行资源调度的方法存在的信息过时、数据不一致的问题；GChord 对 Chord 路由算法进行扩充，使扩充后的算法能更加适用于网格资源的动态性；算法实现过程中主要解决了网格 Overlay 网络的拓扑组织形式、资源的发布方法、资源的更新、资源发现以及路由算法等问题；通过大量的实验证明了该方法的有效性。

多代理技术已经成功地应用于很多研究领域，通过多代理之间的协作，使得在具有不完全知识的情况下处理问题成为可能。多代理的分布性和协同性，适合解决动态的、自治的网格环境中的问题。7.3 节利用多代理的自学习和自组织特性，设计了基于多代理协同计算

的网格资源调度算法 rwAgent，通过多代理间的自组织、自学习和调度管理，很好地解决了网格资源的动态调度问题，同时可以获得全局的负载均衡状态。通过建立数学模型对算法进行了理论分析，最后使用实验验证了对算法设计的可行性。

7.4 节介绍了 DDGrid，新药研发网格的框架结构以及系统平台的实现。它的实现充分利用了本章的研究成果。DDGrid 接受用户从 Web 上提交的药物筛选计算任务，自动取出和计算相关的数据项，进行并行任务划分，然后将这些计算任务映射到网格系统的各计算结点中。DDGrid 利用中国国家网格各计算中心的集群提供的空闲计算力，进行虚拟药物筛选实验计算，在计算结束后，DDGrid 会通知用户通过 Grid Portal 下载计算结果。用户只需提交药物筛选所需的原始文件，DDGrid 便可自动地进行处理，此过程对于用户完全透明。

参考文献

[1] Kwok Y K, Ahmad I. Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors. In: ACM Computing Surveys, Vol. 31, No. 4, 1999. 407-471.

[2] Beck M, Pingali K, Nicolau A. Static Scheduling for Dynamic Dataflow Machines. In: Journal of Parallel Distributed Computing, Vol. 10(4), 1990. 279-288.

[3] Fahringer T. Compile-time Estimation of Communication Costs for Data Parallel Programs. In: Journal of Parallel and Distributed Computing, Vol. 39, 1996. 46-65.

[4] Jiang H, Bhuyan L N, Ghosal D. Approximate Analysis of Multiprocessing Task Graphs. In: the International conference on Parallel Processing, 1990. 228-235.

[5] Marinescu D C, et al. Distributed supercomputing. In: the 10th International Conference on DCS, 1990. 381-387.

[6] Anderson D, Cobb J, et al. Massively distributed computing for SETI. In: Computing in Science & Engineering, 2001.

[7] Anderson D, Cobb J, et al. Seti@home: An experiment in public-resource computing. In: Communications of the ACM 45, 2002. 56-61.

[8] Wu M D, Gajski D. Hypertool: programming aid for message-passing systems. In: IEEE Trans. On Parallel and Distributed Systems. Vol. 1, No. 3, 1990. 330-343.

[9] Cormen T H, Leiserson C E, Rivest R L, Stein C. Introduction to Algorithms. The MIT press, 2001.

[10] Ratnasamy S, Shenker S, Stoica I. Routing Algorithms for DHTs: Some Open Questions. In: the 1st International Workshop on Peer-to-Peer Systems (IPTPS02), 2002.

[11] Gummadi K, Gummadi R, Gribble S, Ratnasamy S, et al. The impact of DHT routing geometry on resilience and proximity. In: SIGCOMM 2003, ACM, 2003. 381-394.

[12] Tyron Stading, Petros Maniatis, Mary Baker. Peer-to-Peer Caching Schemes to Address Flash Crowds. In: 1st International Workshop on Peer-to-Peer Systems, 2002.

[13] Chen Y, Rand H K, Kubiatowicz J D. Dynamic Replica Placement for Scalable Content Delivery. In: 1st International Workshop on Peer-to-Peer Systems, 2002.

[14] Ananth R, Lakshminarayanan K, Surana S, Karp R, Ion S. Load Balancing in Structured P2P Systems. In: IPTPS2003, 2003.

[15] Xu Z, Zhang Z. Building Low-maintenance Expressways for P2P Systems. Internet Systems and Storage Laboratory, HP Laboratories Palo Alto, HPL-2002-41, 2002.

[16] Garcés-Erice L, Ross K W, Biersack E W, Felber P A, Urvoy-Keller G. Topology-Centric Look-Up Service. In: COST264/ACM Fifth International Workshop on Networked Group Communications

(NGC),2003. 58-69.

[17] Castro M, Druschel P, Hu Y, Rowstron A. Topology-aware Routing in Structured Peer-to-Peer Overlay Networks. MSR-TR,2002.

[18] Ratnasamy S,Francis P,Handley M,Karp R,Schenker S. A Scalable Content-Addressable Network. In: SIGCOMM 2001,ACM,2001. 161-172.

[19] Ratnasamy S, Handley M, Karp R, Shenker S. Application Level Multicast Using Content-Addressable Networks. In: the Third International Workshop on Networked Group Communication (NGC),Lecture Notes in Computer Science,Vol. 2233,2001. 14-29.

[20] Saroiu S,Gummadi P K,Gribble S D. A Measurement Study of Peer-to-Peer File Sharing Systems. In: Multimedia Computing and Networking 2002 (MMCN),2002.

[21] Zhu Y,Wang H,Hu Y. In: the 16th International Conference on Parallel and Distributed Computing Systems (PDCS03),2003.

[22] Krishnamurthy B,Wang J. On Network-Aware Clustering of Web Clients. ACM SIGCOMM,2001.

[23] Stoica I,Morris R,Karger D,Kaashoek M F,Balakrishnan H. Chord: A scalable peer-to-peer lookup service for Internet applications. In: SIGCOMM 2001,ACM,2001. 149-160.

[24] Zegura E W,Calvert K,Bhattacharjee S. How to model an Internetwork. In: IEEE INFOCOM 1996, 1996.

[25] http://www. w3. org/2002/ws/desc. WSDL Specification version 1. 2.

[26] Aversa L, Bestavros A. Load Balancing a Cluster of Web Servers Using Distributed Packet Rewriting. In: IEEE International Performance,Computing,and Communications Conference,2000.

[27] Andresen D, Yang T, Holmedahl V, Ibarra O H. SWEB: Towards a Scalable WWWServer on MultiComputers. In: IEEE International Symposium on Parallel Processing,1996.

[28] Cardellini V,Colajanni M, Yu P S. Redirection Algorithms for Load Sharing in Distributed Web Server Systems. In: ICDCS,1999.

[29] Castro M,Dwyer M, Rumsewicz M. Load balancing and control for distributed World Wide Web servers. In: the IEEE International Conference on Control Applications,1999.

[30] Dandamudi S. Performance Impact of Scheduling Discipline on Adaptive Load Sharing in Homogeneous Distributed Systems. In: ICDCS,1995.

[31] Lu C,Lau S M. An Adaptive Load Balancing Algorithm for Heterogeneous Distributed Systems with Multiple Task Classes. In: ICDCS,1996.

[32] Petri S,Langendorfer H. Load Balancing and Fault Tolerance in Workstation Clusters - Migrating Groups of Communicating Processes. In: Operating Systems Review,Vol. 29(4),1995. 25-36.

[33] Frey J,Tannenbaum T,Foster I,Livny M,Tuecke S. Condor-G: A Computation Management Agent for Multi-Institutional Grids. In: the Tenth IEEE Symposium on High Performance Distributed Computing,2001.

[34] Abramson D,Giddy J, Kotler L. High performance parametric modeling with Nimrod/G: killer application for the global grid. In: the 14th International Parallel and Distributed Processing Symposium,2000.

[35] BioMagResBank. A repository for data from NMR spectroscopy on proteins, peptides, and nucleic acids. http://www. bmrb. wisc. edu/,2004.

[36] Blast project. http://www. ncbi. nlm. nih. gov/BLAST.

[37] Stone P, Veloso M. Multiagent System: A Survey from a Machine Learning Perspective. In: Autonomous Robots,Vol. 3(8),2000. 345-383.

[38] Agha G. Actors: A Model of Concurrent Computation in Distributed Systems. The MIT Press: Cambridge,MA,1986.

[39] Schlenoff C, Ivester R, Knutilla A. A Robust Ontology for Manufacturing Systems Integration. In: Proceedings of the 2nd International Conference on Engineering Design and Automation, 1998.
[40] Bussmann S. An agent-oriented architecture for holonic manufacturing control. In: 1st International Workshop on intelligent Manufacturing System (EPFL), 1998. 1-12.
[41] Luck M, McBurney P, Preist C. Agent technology: Enabling Next Generation Computing. In: AgentLink, 2003.
[42] Ian Foster, Nicholas R. Jennings, Carl Kesselman, Brain Meets Brawn. Why Grid and Agents Need Each Other. In: the Third International Joint Conference on Autonomous Agents and Multiagent Systems, Vol. 1, 2004.
[43] Galstyan A, Czajkowski K, Lerman K. Resource Allocation in the Grid Using Reinforcement Learning. In: International Conference on Autonomous Agents and Multiagent Systems, 2004.
[44] Wolski R, Brevik J, Plank J, Bryan T. Grid Resource Allocation and Control Using Computational Economies. In: Berman, F, Fox, G. and Hey, T. eds. Grid Computing: Making the Global Infrastructure a Reality, Wiley and Sons, 2003. 747-772.
[45] Montresor A, Meling Messor H. Load-balancing through a swarm of autonomous agents. Technical Report, Dept. of Computer Science, University of Bologna, UBLCS-02-08, 2002.
[46] Bunruangses M, Poompattanapong W, Banyatnoparat P, Piyatamrong B. QoS Multi-Agent Applied for Grid Service Management. In: the 3rd International Symposium on Information and Communication Technologies. 2004.
[47] Babaoglu O, Meling H, Montresor A. Anthill: A framework for the development of agent-based peer-to-peer system. In: the 22th Int. Conf. On Distributed Computing Systems, 2002.
[48] Cao J, Augusting S. Self-Organizing Agents for Grid Load Balancing. In: fifth IEEE/ACM International Workshop on Grid Computing, 2004. 388-395.
[49] Spooner D P, Cao J. Agent-Based Grid Load-Balancing.
[50] The jxta homepage. Project JXTA: An Open, Innovative Collaboration. http://www.jxta.org/project/www/white_papers.html.
[51] Lerman K, Shehory O. Coalition formation for large-scale electronic markets. In: the Internal Conference on Multi-Agent Systems, 2000.
[52] Iennings, Campos I R. Towards a Social Level Characterization of Socially Responsible Agents. In: Software Engineering, IEEE Computer Society, Vol. 144(1), 1997. 11-25.
[53] The Intel Philanthropic Peer-to-Peer Program. http://www.intel.com/cure.
[54] Foster I, Kesselman C, Tuecke S. The Anatomy of the Grid: Enabling Scalable Virtual Organizations, In: Journal of Supercomputer Applications and High Performance Computing, 2001.

CHAPTER 8

第8章 云计算文件系统设计与实现

8.1 引言

8.1.1 云计算概述

目前，云计算是个很热的话题，可以说人类将要进入云计算的时代。人们如此重视云计算的原因，一方面是云计算这种商业模式孕育着巨大的商业机遇，另一方面，云计算作为互联网资源新的配置方式，它将改变整个互联网的价值链。云计算是网格计算(Grid Computing)、分布式计算(Distributed Computing)、并行计算(Parallel Computing)、效用计算(Utility Computing)网络存储(Network Storage Technologies)、虚拟化(Virtualization)、负载均衡(Load Balance)等传统计算机技术和网络技术发展融合的产物。云计算提供无所不在、无时不在的互联网服务，它将彻底改变人们使用硬件、软件、数据的方式，使人们摆脱传统的信息技术使用方式的束缚，能够更加关注信息和内容的本身，而不是其他。

云计算的出现将改变人们对IT系统(包括硬件和软件)的使用方式。第一，用户没有必要购买、安装、维护自己的IT设备，只要通过网络连接就可以租用网络上提供的应用程序、计算资源和存储资源服务。第二，云计算将改变人们使用软件的方式，用户现在安装在个人电脑上的所有软件都可以通过网络方式获得在线软件服务，比如：个人可以通过网络租用Office软件服务，企业可以通过网络租用CRM软件服务。第三，云计算将改变人们信息访问的方式，用户通过终端就可以访问“云”中的信息，等等。

云计算目前还没有明确的定义，Wikipedia关于云计算的定义为“云计算是网格计算下的一种新的标签，它使用公用计算或其他方法来共享计算资源。云计算是依靠本机服务器或个人设备来处理用户应用程序之外的另一种选择”。News Blog认为“云计算是一种将硬件与软件外包给因特网服务提供商的概念”。

云计算有狭义云计算和广义云计算之分。狭义云计算指的是 IT 基础设施的交付和使用模式，即通过网络以按需、易扩展的方式获得所需的资源（硬件、平台、软件）。“云”中的资源在使用者看来是可以无限扩展的，并且可以随时获取，按需使用，随时扩展，按使用付费。广义云计算指的是服务的交付和使用模式，即通过网络以按需、易扩展的方式获得所需的服务。这种服务可以是 IT 和软件、互联网相关的，也可以是任意其他的服务。

8.1.2 云计算关键技术

云计算的主要关键技术如下[1]：

1. 虚拟化（Virtualization）

虚拟化是云计算的核心特征之一。虚拟化是指计算机元件在虚拟的基础上而不是真实的基础上运行。虚拟化技术可以扩大硬件的容量，简化软件的重新配置过程。CPU 的虚拟化技术可以单 CPU 模拟多 CPU 并行，允许一个平台同时运行多个操作系统，并且应用程序都可以在相互独立的空间内运行而互不影响，从而显著提高计算机的工作效率。

2. IaaS/PaaS/SaaS 服务模式

云计算中用户所需要的任何东西都以服务形式体现，服务类型包括计算服务、网络服务、软件服务、平台服务、存储服务等。

IaaS 是将基础设施（主要为计算资源、网络资源和存储资源）作为服务出租，PaaS 是 IaaS 加上一个用于给定应用的定制软件栈，可描述为一个完整的虚拟平台，它包括操作系统和围绕特定应用的必需的服务。SaaS 是一种基于互联网提供软件服务的应用模式，是管理软件的发展趋势，也是云计算部署的最佳实践。

3. 并行计算和并行算法

并行计算（Parallel Computing）是指同时使用多种计算资源解决计算问题的过程。为执行并行计算，计算资源应包括一台配有多处理机（并行处理）的计算机、一个与网络相连的计算机专有编号，或者两者结合使用。并行计算的主要目的是快速解决大型且复杂的计算问题。此外还包括利用非本地资源，节约成本，即使用多个“廉价”计算资源取代大型计算机，同时克服单个计算机上存在的存储器限制。

并行计算科学中主要研究的是空间上的并行问题。从程序和算法设计人员的角度来看，并行计算又可分为数据并行和任务并行。一般来说，因为数据并行主要是将一个大任务化解成相同的各个子任务，比任务并行要容易处理。

空间上的并行导致了两类并行机的产生，按照 Flynn 的说法分为：单指令流多数据流（SIMD）和多指令流多数据流（MIMD）。常用的串行机也叫做单指令流单数据流（SISD）。MIMD 类的机器又可分为以下常见的五类：并行向量处理机（PVP）、对称多处理机（SMP）、大规模并行处理机（MPP）、工作站机群（COW）、分布式共享存储处理机（DSM）。

4. 面向服务的体系结构

SOA（Service-Oriented Architecture，面向服务架构）是指为了解决在 Internet 环境下业务集成的需要，通过连接能完成特定任务的独立功能实体实现的一种软件系统架构。SOA 是一个组件模型，它将应用程序的不同功能单元（称为服务）通过这些服务之间定义良

好的接口和契约联系起来。接口是采用中立的方式进行定义的,它应该独立于实现服务的硬件平台、操作系统和编程语言。这使得构建在各种这样的系统中的服务可以以一种统一和通用的方式进行交互。概括地说,SOA 具有以下特点:松耦合性、位置透明和协议的独立性。松耦合性要求 SOA 架构中的不同服务之间应该保持一种松耦合的关系,也就是应该保持一种相对独立无依赖的关系;位置透明性要求 SOA 系统中的所有服务对于它们的调用者来说都是位置透明的,也就是说每个服务的调用者只需要知道他们调用的是哪一个服务,但并不需要知道所调用服务的物理位置在哪里;而协议独立性要求每一个服务都可以通过不同的协议来调用。这样基于 SOA 建立的系统可以确保整个系统的松散耦合和灵活性,这将为企业未来的业务需求扩张的打下良好基础,也就是说,SOA 可以提供很好的业务灵活性。可以迅速有效地响应业务需求的变化。

在 SOA 出现之前,软件包通常被编写为独立的软件,即在一个完整的软件包中将许多应用程序功能整合在一起。该种开发模式将其业务逻辑和 IT 实现捆绑在一起来实现,当业务需求发生变化时修改代码不可避免,这会造成系统灵活性差,并增加维护系统的成本,使重用应用程序变得困难。SOA 旨在将单个应用程序功能彼此分开,以便这些功能可以单独用作单个的应用程序功能或“组件”。这些组件可以用于在企业内部创建各种其他的应用程序,或者对外公开供其他应用程序使用。SOA 强调重用,但是相对于传统的代码重用、对象重用和构件重用,SOA 的重用粒度更粗。SOA 的重用在于业务级的应用,即服务的重用。SOA 架构的主要优势有:

(1) 减少企业 IT 支出。SOA 架构实现了各系统服务间的松散结合,这种方法降低新业务整合的复杂度,从而减少 IT 开发成本和开发时间。

(2) 提高资产再利用。SOA 建设属于 IT 基础设施方面的范畴,SOA 架构可实现对现有系统(服务)的再利用,并且 SOA 的经济性将随公司创造并再重复利用的服务数量的增加而增加。

(3) 降低业务风险。SOA 的 IT 架构可以更快、更好地适应业务的变化,降低业务风险,并且提供业务流程的整体可见性,反过来改进业务流程,降低业务风险。

SOA 是一种架构,云计算是基于某个架构之上的结果,云计算有很多方面与 SOA 有交叉。SOA 治理(也称为服务治理),指的是保证一些核心功能在整个生命周期的开发、安全、性能以及其他策略上的一致。随着云计算和 SOA 不断融合,对治理策略、治理技术的需求日益迫切。

5. 云存储

云存储是在云计算(Cloud Computing)概念上延伸和发展出来的一个新的概念,当云计算系统运算和处理的核心是大量数据的存储和管理时,云计算系统中就需要配置大量的存储设备,那么云计算系统就转变成为一个云存储系统,所以云存储是一个以数据存储和管理为核心的云计算系统。

云存储中的存储设备数量庞大且分布多在不同地域,如何实现不同厂商、不同型号甚至于不同类型(如 FC 存储和 IP 存储)的多台设备之间的逻辑卷管理、存储虚拟化管理和多链路冗余管理将会是一个巨大的难题,这个问题得不到解决,存储设备就会是整个云存储系统的性能瓶颈,结构上也无法形成一个整体,而且还会带来后期容量和性能扩展难等问题。

云存储中的另外一个问题就是存储设备运营管理问题。对于云存储的运营单位来讲,

必须要通过切实可行和有效的手段来解决集中管理难、状态监控难、故障维护难、人力成本高等问题。因此，云存储必须要具有一个高效的类似于网络管理软件一样的集中管理平台，可实现云存储系统中设有存储设备、服务器和网络设备的集中管理与状态监控。

6. 云计算服务与业务流程管理的融合

BPM(Business Process Management，业务流程管理)是利用工作流技术，从业务流程的角度对企业进行全方位的管理，将业务逻辑和业务执行分开，使信息系统能够快速地适应流程的变化，可减少业务需求与 IT 系统间的失配，降低运营成本，提高 IT 系统的机动性和灵活性；也正是由于业务逻辑与代码实现的分离，从而实现对大部分环境变化可通过修改规则库即可满足，提高了程序的柔性。

在云计算中，一切皆为服务，服务是云计算平台中的接口，采用云计算平台的服务架构将众多企业的业务功能均以服务的形式发布出来，而云计算中服务具有标准接口以及松散耦合等特点，这使得企业的 BPM 能够方便地通过云计算服务的组合来灵活构建自己企业的业务流程以适应新的业务需求。BPM 和云计算服务两者是相辅相成的，没有 BPM 对流程进行深入分析和管理，就不会真正知道应该整合哪些服务以方便重用，此外，BPM 可对业务流程进行深入分析从而更好地改善企业的业务流程，提高效率。

8.2 云存储

8.2.1 云存储系统架构

云存储是指通过网格技术、集群(Cluster)应用或分布式文件系统等功能，将网络中大量各种不同类型的存储设备通过应用软件集合起来协同工作，共同对外提供数据存储和业务访问功能的一个服务系统。

与传统的存储设备相比，云存储不仅仅是一个硬件，而是一个网络设备、存储设备、服务器、应用软件、公用访问接口、接入网和客户端程序等多个部分组成的复杂系统。各部分以存储设备为核心，通过应用软件来对外提供数据存储和业务访问服务。云存储系统的结构模型由四层组成[2]，如图 8-1 所示。

1. 存储层

存储层是云存储最基础的部分。存储设备可以是 FC 光纤通道存储设备，可以是 NAS 和 iSCSI 等 IP 存储设备，也可以是 SCSI 或 SAS 等 DAS 存储设备。云存储中的存储设备往往数量庞大且分布于不同地域，彼此之间通过广域网、互联网或者 FC 光纤通道网络连接在一起。

存储设备之上是一个统一存储设备管理系统，可以实现存储设备的逻辑虚拟化管理、多链路冗余管理，以及硬件设备的状态监控和故障维护。

2. 基础管理层

基础管理层是云存储最核心的部分，也是云存储中最难以实现的部分。基础管理层通过集群、分布式文件系统和网格计算等技术，实现云存储中多个存储设备之间的协同工作，使多个的存储设备可以对外提供同一种服务，并提供更大、更强、更好的数据访问性能。

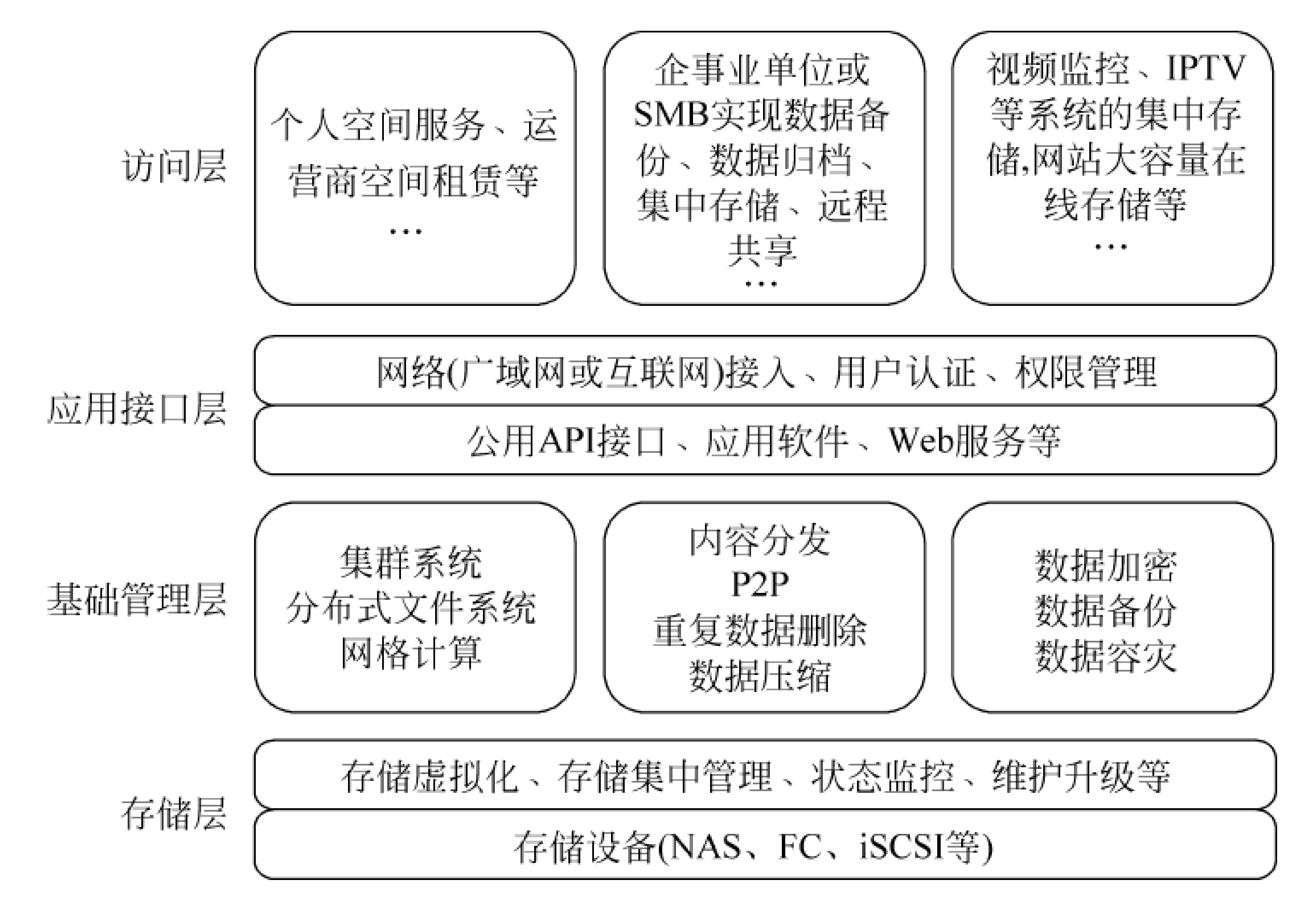


图 8-1　云存储系统的结构模型

CDN 内容分发系统、数据加密技术保证云存储中的数据不会被未授权的用户所访问，同时，通过各种数据备份和容灾技术与措施可以保证云存储中的数据不会丢失，保证云存储自身的安全和稳定。

3. 应用接口层

应用接口层是云存储最灵活多变的部分。不同的云存储运营单位可以根据实际业务类型，开发不同的应用服务接口，提供不同的应用服务。比如视频监控应用平台、IPTV 和视频点播应用平台、网络硬盘引用平台，远程数据备份应用平台等。

4. 访问层

任何一个授权用户都可以通过标准的公用应用接口登录云存储系统，享受云存储服务。云存储运营单位不同，云存储提供的访问类型和访问手段也不同。

8.2.2　云存储的优势

与传统的购买存储设备和部署存储软件相比，云存储方式存在以下优点[3,4]：

1. 成本低、见效快

传统的购买存储设备或软件定制方式下，企业通常需要投入大量资金购置硬件设备、搭建平台。在云存储方式下，企业除了配置必要的终端设备接收存储服务外，不需要投入额外的资金来搭建平台。企业只需按用户数分期租用云存储服务，从而规避了一次性投资的风险，降低了使用、维护成本。

2. 易于管理

在传统方式下，企业需要配备专业的 IT 人员进行系统的维护，由此带来技术和资金成本。云存储模式下，维护工作以及系统的更新升级都由云存储服务提供商完成，企业能够以最低的成本享受到最新、最专业的服务。存储的管理非常复杂，不同存储厂商有不同的管理

界面，数据中心人员经常需要面对不同的存储产品，这种情况下，了解每个存储的使用状况（容量、负载等），变得非常复杂。对云存储来说，再多的存储服务器，在管理人员眼中，只是一台存储，管理人员只要在整体硬盘容量快用完时，采购服务器即可，每台存储服务器的使用状况，都可以在一个管理界面上看到。

3. 方式灵活

传统的购买和定制模式下，一旦完成资金的一次性投入，系统无法在后续使用中动态调整。随着设备的更新换代，落后的硬件平台难以处置；随着业务需求的不断变化，软件需要不断地更新升级甚至重构来与之相适应，导致维护成本高昂，很容易发展到不可控的程度。还有，传统的存储系统升级时，需要把旧的存储设备文件备份出来后，停机，换上新的存储设备，这会导致服务的停止。

而云存储方式一般按照客户数、使用时间、服务项目进行收费。企业可以根据业务需求变化、人员增减、资金承受能力，随时调整其租用服务方式，真正做到“按需使用”。云存储并不单独依赖一台存储服务器，因此存储服务器硬件的更新、升级并不会影响存储服务的提供，系统会将旧的存储服务器上的文件迁移到别的存储服务器，等新的存储服务器上线后，文件会再迁移回来，升级方便。

4. 可靠性高

很多种原因的硬件损坏会导致服务的停止，例如硬盘、主板、电源、网卡等，虽然针对这些弱点，管理人员可以找到替代方案，例如建立一个全冗余的环境（电源、网络、盘阵等），但是这样的成本太高而且工作非常繁复。

云存储透过将文件复制并且存在不同的服务器，解决了这个潜在的硬件损坏的难题。云存储知道文件存放的位置，在硬件发生损坏时，系统会自动将读写指令导向存放在另一台存储服务器上的文件，保持服务的继续。

以上也是云存储会受欢迎的理由，是用户面对呈爆炸性成长的海量数据时最好的选择。

8.3 云计算文件系统概述

云计算中的文件系统主要是分布式文件系统，分布式文件系统主要分布在多个计算机结点上，每个结点只会直接存取整个文件系统的一部分。一般来说，分布式文件系统均包括复制和容错功能，从而保障系统在某些结点出错的情况下仍能够继续正常工作，而不会出现数据丢失的情况。

云计算的分布式文件系统目前主要有 GFS(Google File System)，下面将先介绍 GFS，然后介绍 Hadoop，以及一个基于云计算的文件系统。

8.3.1 GFS

GFS 是一个可扩展的分布式文件系统，用于大型的、分布式的、对大量数据进行访问的应用。它可运行于廉价的普通硬件上，并提供强大的容错功能。它可以给大量的用户提供总体性能较高的服务[5]。

GFS 与以往的文件系统的不同在于：

（1）允许组件发生错误。考虑到分布式文件系统是由很多存储的机器构成，而这些机器是由廉价的普通部件组成并被大量的客户机访问。组件的数量和质量使得一些机器随时都有可能无法工作并且有一部分还可能无法恢复。所以实时地监控、错误检测、容错、自动恢复对系统来说必不可少。

（2）大块文件和小块文件都支持。GFS 中达几个 GB 的文件是很平常的。每个文件通常包含很多应用对象。当经常要处理快速增长的、包含数以万计的对象、长度达 TB 的数据集时，将很难管理成千上万的 KB 规模的文件块，即使底层文件系统提供支持。因此，设计中操作的参数、块的大小必须要重新考虑。

（3）文件修改主要是发生于添加新数据，而不是改变已存在的数据。一旦写完，文件就只可读，有很多数据都有这些特性。一些数据可能组成一个大仓库以供数据分析程序扫描。有些是运行中的程序连续产生的数据流。有些是档案性质的数据，有些是在某个机器上产生、在另外一个机器上处理的中间数据。由于这些对大型文件的访问方式，添加操作成为性能优化和原子性保证的焦点。而在客户机中缓存数据块则失去了吸引力。

（4）工作量主要由两种读操作构成：对大量数据的流方式的读操作和对少量数据的随机方式的读操作。在前一种读操作中，可能要读几百 KB，通常达 1MB 和更多。来自同一个客户的连续操作通常会读文件的一个连续的区域。随机的读操作通常在一个随机的偏移处读几个 KB。性能敏感的应用程序通常将对少量数据的读操作进行分类并进行批处理以使得读操作稳定地向前推进，而不要让它来来回回地读。

（5）工作量还包含许多对大量数据进行的、连续的、向文件添加数据的写操作。所写的数据的规模和读相似。一旦写完，文件很少改动。在随机位置对少量数据的写操作也支持，但不必非常高效。

（6）系统必须高效地实现定义完好的大量客户同时向同一个文件的添加操作的语义。

8.3.2　Hadoop

Hadoop 文件系统是一个能兼容普通硬件环境的分布式文件系统，和现有的分布式文件系统不同的地方是 Hadoop 更注重容错性和兼容廉价的硬件设备，这样做是为了用很小的预算甚至直接利用现有机器就实现大流量和大数据量的读取。Hadoop 使用了 POSIX 的设计来实现对文件系统文件流的读取。HDFS（Hadoop FileSystem）原来是 Apache Nutch 搜索引擎（从 Lucene 发展而来）开发的一个部分，后来独立出来作为一个 Apache 子项目[6~10]。

1. Hadoop 体系结构

Hadoop 文件系统是主从架构，一个 Hadoop 文件系统由唯一一个目录结点（NameNode）和多个数据结点（DataNode）组成。Hadoop 文件系统对外表现为一个普通的文件系统，用户可以用文件名存储和访问文件，而实际上文件是被分成不同的数据块，这些数据块存储在数据结点上。

目录结点是集群里面的主结点，负责文件名的维护管理，也是客户端访问文件的入口。文件名的维护包括文件和目录的创建、删除、重命名等。同时也管理数据块和数据结点的映射关系，客户端需要访问目录结点才能知道一个文件的所有数据块都保存在哪些数据结点上。

数据结点一般就是集群里面的一台机器，负责数据的存储和读取。在写入时，由目录结点分配数据块的保存，然后客户端直接写到对应的数据结点。在读取时，当客户端从目录结点获得数据块的映射关系后，就会直接到对应的数据结点读取数据。数据结点也要根据目录结点的命令创建、删除数据块和冗余复制。一个典型的 Hadoop 文件系统集群部署，是由一台性能较好的机器运行目录结点，而集群里面的其他机器每台上面运行一个数据结点。当然一个机器可以运行任意多个数据结点，甚至目录结点和数据结点一起运行，不过这种模式在正式的应用部署中很少使用。

唯一的目录结点的设计大大简化了整个体系结构，目录结点负责 Hadoop 文件系统里所有元数据的仲裁和存储。这样的设计使数据不会脱离目录结点的控制。Hadoop 文件系统的整体结构见图 8-2。

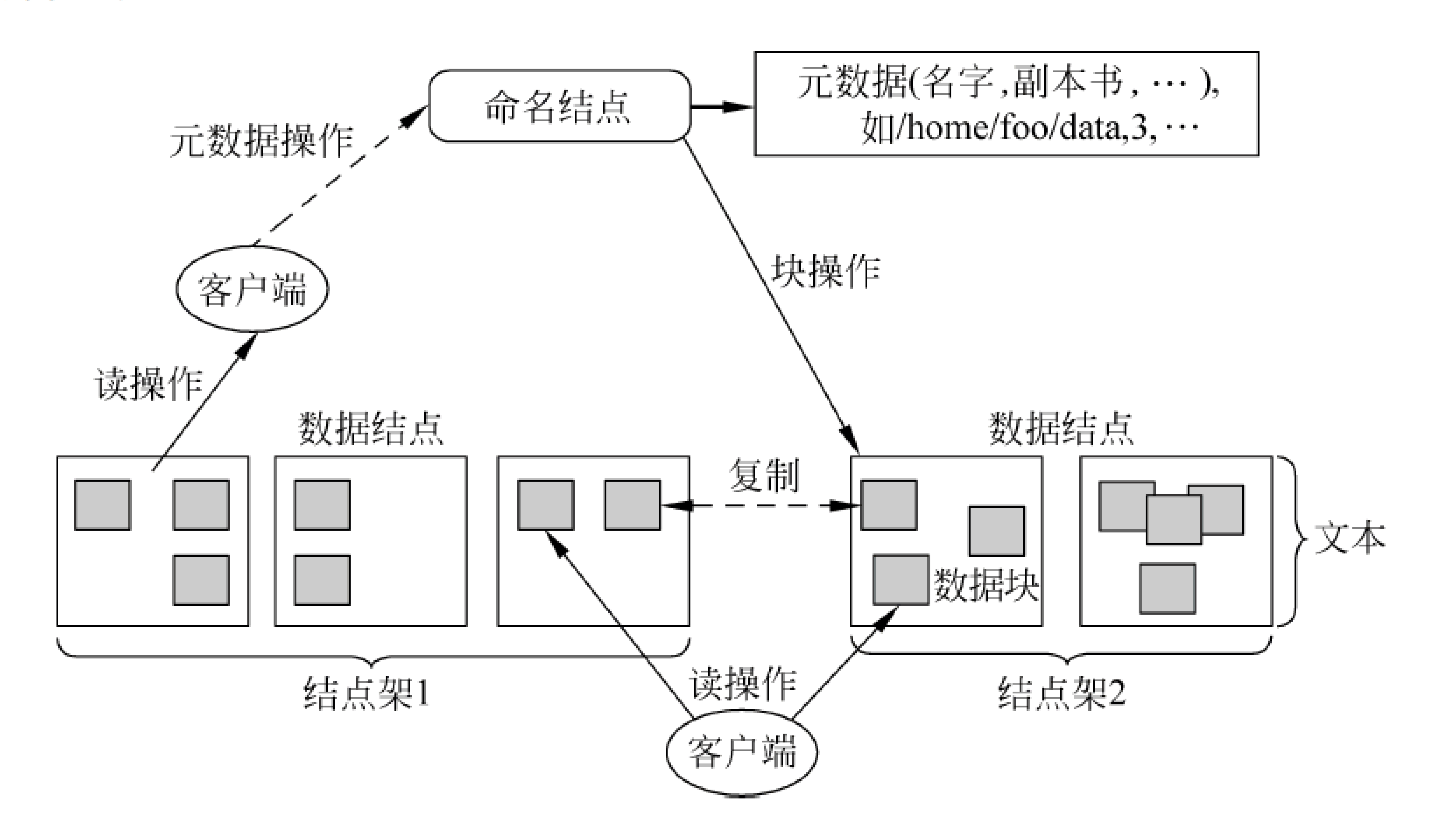


图 8-2　Hadoop 文件系统的整体结构

Hadoop 文件系统使用的是传统的分级文件体系，客户端程序可以创建目录并且在目录里面保存文件，类似于现在一般的文件系统。Hadoop 允许用户创建、删除文件，在目录间转移文件，重命名文件等，但是还没有实现磁盘配额和文件访问权限等功能，也不支持文件的硬连接和软连接(快捷方式)，这些功能在短期内不会实现。

目录结点主要负责存储和管理整个文件系统的命名空间，应用程序可以指定某一个文件需要在 Hadoop 文件系统中冗余多少份，这个在 Hadoop 中称为冗余因素，保存在目录结点里面。

2. Hadoop 存储原理

Hadoop 文件系统是为了大文件的可靠保存而设计的，一个文件被划分成一连串的数据块，除了文件的最后一块以外其他所有的数据块都是固定大小的，为了数据容错性，每一个数据块都会被冗余存储，而每个文件的块大小和冗余因素都是可以设置的，程序可以设置文件的数据块要被复制多少份，而且这个冗余因素除了可以在创建的时候指定，还可以在之后改变。在 Hadoop 文件系统里面文件只会被写入一次，并且任何时间只会有一个程序在写入这个文件。

目录结点是根据数据块的冗余状况来作出处理决策的，数据结点会定期发送一个存在信号(Heartbeat)和数据块列表给目录结点，存在信号使目录结点认为该数据结点还是有效

的，而数据块列表包括了该数据结点上面的所有数据块编号。

8.3.3　基于云计算的文件系统

下面是一个面向数据密集型云计算应用的分布式文件系统(Cloud based File System，CFS)案例，该系统的主要功能和应用集中于文件的分布式存储和读写。整个系统由多个Master结点、数据结点和客户端组成。它具有强容错性和可扩展性，屏蔽操作系统和硬件的异构性。

1. 系统设计

该分布式文件系统由客户端、调度模块、存储模块三个角色组成。其中客户端主要为用户提供文件的上传、下载、查询等功能；调度模块部分主要实现文件跟踪功能，为文件的存储进行调度；存储模块部分用来存储文件，完成文件的存储、同步，并提取存取接口。整个系统的框架结构如图8-3所示。

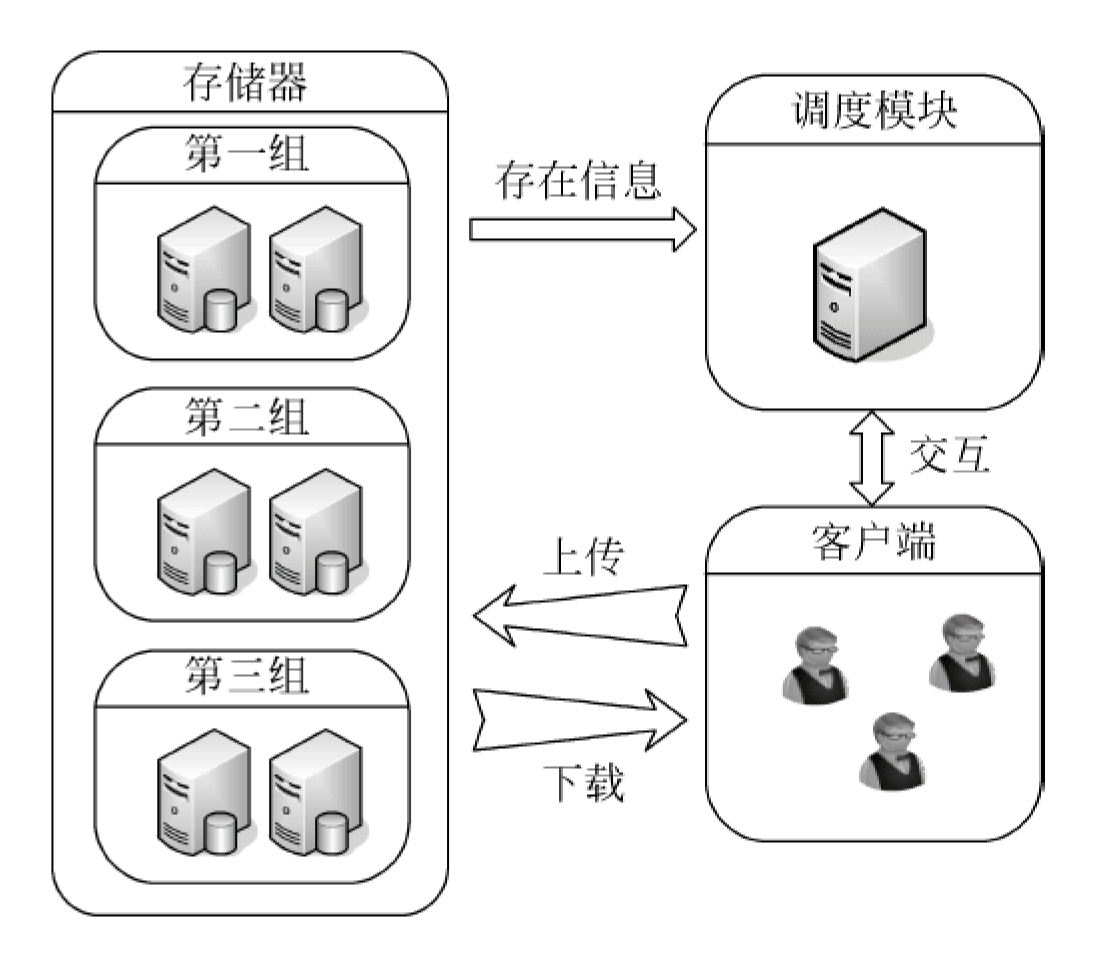


图 8-3　云计算的文件系统 CFS 的框架结构

为了满足动态的网络环境，调度模块和存储模块都由多台机器组成。若主调度模块发生宕机，则由其他的调度模块承担跟踪任务，调度模块之间定时交互整个文件系统的信息。若存储模块发生宕机，则当用户提交申请时返回给同组内的其他机器，以便使用其副本。存储模块采用分组的方式进行组织，整个文件系统由多个组组成，同一组由多台存储服务器组成。同一组内的机器存储的内容是相同的，保证了同一文件具有足够的副本，在某个结点发生宕机后，可由同组的其他机器提供备份。需要注意的是，这里的组只是一个逻辑概念。同一组的存储服务器在调度模块中由一个链表组织起来，若某个服务器发生宕机，则调度模块将结点从该组的链表中删除。

(1) 上传文件功能。客户端询问调度模块，提出要上传文件的请求；调度模块返回一台可用的存储服务器地址；客户端直接与存储模块通信完成文件上传功能。

(2) 下载文件功能。客户端向调度模块提出下载文件的请求并提供文件名；调度模块返回可供下载的存储服务器地址；客户端直接与存储模块通信完成文件下载功能。

(3) 查询文件功能。客户端向调度模块提供待查询文件名或文件名关键字；调度模块返回所有可用的存储服务器列表。

2. 模块设计

CFS 系统目前主要包括调度模块、存储模块、客户端。

1）调度模块

调度模块是整个分布式系统的管理结点，与客户端和存储模块相连，负责接收来自客户端的服务请求以及与存储模块的数据交互。主要功能有提供服务接口、检索文件列表、查找存储服务器、读写元数据、数据备份等，其结构如图 8-4 所示。

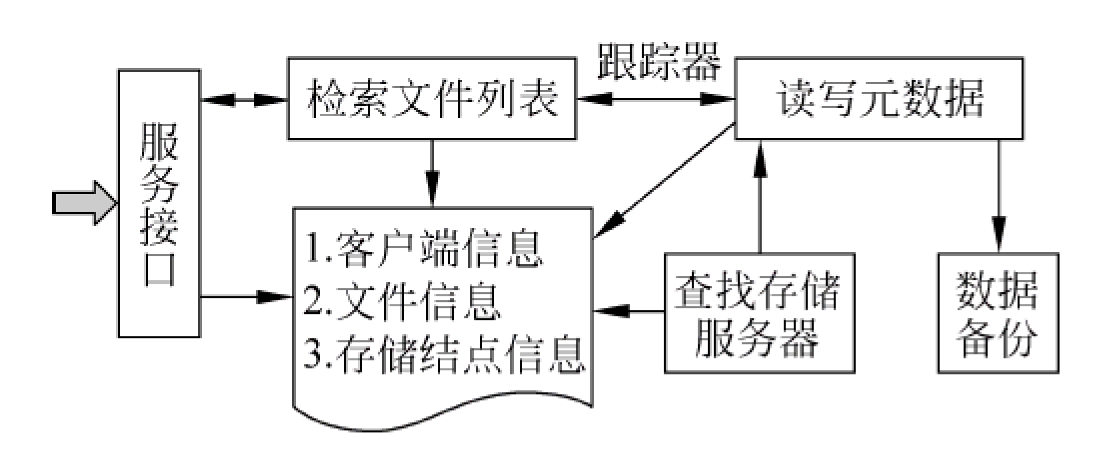


图 8-4 调度模块结构

服务接口指的是调度模块接受来自客户端的请求服务，通过系统中用户信息检查当前用户是否为注册客户，并根据不同请求执行相应操作。检索文件列表指根据客户端发送过来的文件名，在文件信息中检索。若检索到则返回该文件在存储模块中的存储位置，否则返回错误信息。在系统中采用参考文献[11]中的算法来产生字典序树，这种算法实现简单，算法时间复杂度小，存储空间小。通过字典序树，系统可以很方便地进行文件检索。读写元数据是指当客户端查询文件信息时，调度模块读出相应文件的元数据传送给客户端；当用户往存储模块写文件时，调度模块首先查找存储信息，然后返回给客户端相应的存储服务器 IP 地址，最后调度模块接受和保存来自存储模块的文件存储数据，并把该元数据写入到系统。这里的元数据是定义的一组数据结构，包括文件存储地址的 IP，路径等信息。查找服务器是指寻找存储模块中可用的存储服务器。为了减少查找可用服务器中的存储空间和当前状态的额外开销，调度模块与存储模块中的一个代表进行信息的交互，称为代表服务器。代表服务器每隔一段时间记录来自其他组存储服务器的存储状态，并把该记录信息发送给其他存储服务器。若当前代表服务器宕机，则让该组中的下一个存储服务器作为代表服务器，并与调度模块进行联系。

此外，在设计调度模块的过程中，也考虑了负载均衡问题。调度模块在选择存储空间时，以每个组中存储空间最小的存储服务器为标准，设该存储服务器中的已使用存储空间 s_{used} 与总存储空间 s_{total} 的比值 $s_{utility}=s_{used}/s_{total}$，并以此来确定最终选取某个存储服务器。在存储资源分配算法中，将组按 $s_{utility}$ 值从小到大排序，每次选 $s_{utility}$ 最小的组给文件，若文件需分块存储，则可选择按 $s_{utility}$ 值从小到大排序的前 k 个（top k）组来存储。此外，可以根据加入文件本身的大小和系统当前的状态等因素改进分配算法。

为了防止调度模块失效，增加了冗余服务器，一旦当前服务器失效则由另外的服务器作为调度模块提供服务。因此先前调度模块必须把文件列表在周期内传送给冗余服务器，在当前服务器失效后管理结点仍可达到无缝切换。

2）存储模块

存储模块是分布式文件系统的存储单元，文件系统中的所有文件都保存在各个存储服

务器的不同组中。存储模块需和调度模块交互文件系统的状态信息，直接和客户端进行文件的传输。存储模块的主要功能分为文件存储、存在信息传输、服务接口等，其主要结构如图 8-5 所示。

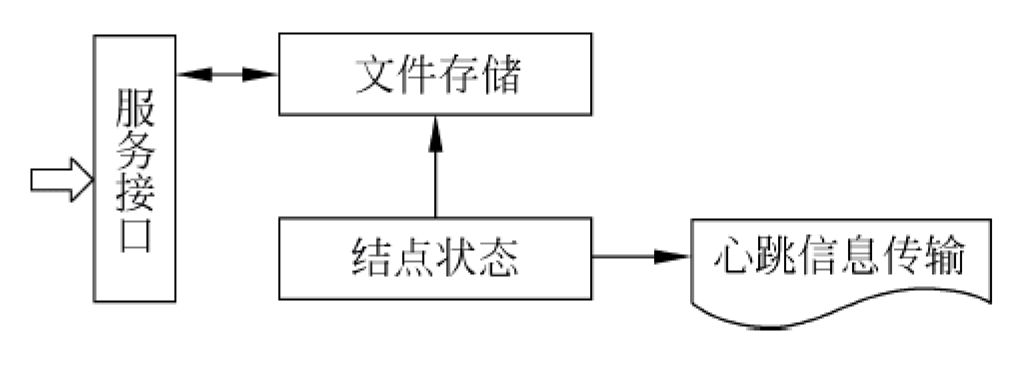


图 8-5　存储模块结构

文件存储是存储模块最主要的功能。分布式文件系统的所有文件都存储在各个存储模块结点中。由于分布式文件系统是由大量的存储结点组成的，每一组结点都有失效的可能，因此必须为每一个文件块准备一定数量的副本。在文件系统中引入了组的概念，在这里组是一个逻辑概念，每个文件块都会在同组的其他服务器上放置副本以满足冗余性。需要注意的是这里文件被切分成很多个 32MB 大小的文件块，每个文件块利用 hash 函数的方法选取一个组冗余存放，同时把键值存储在调度模块中，在查找文件时根据键值构成一一映射的关系，该方法实现简单且有较低的时间复杂度。

存在信息传输是为了解决分布式文件系统高失效性的特点而设计的。当某个存储结点的机器宕机时，分布式文件系统应当在一定时间内检测到该意外情况并立即启用该存储结点文件的其他副本，避免文件丢失。因此，这里设计让代表服务器每隔一定的时延向调度模块发送自己的存在信息，并报告自身健康状况。若在规定的时延后调度模块仍没有收到某存储服务器发送的存在信息，调度模块则认为该服务器失效，并立即启用同一组内其他服务器上的副本。除此之外，代表服务器还会将自己存储的文件列表传递给调度模块，以供其更新整个文件系统的存储列表。

服务接口是指存储模块直接与客户端交互，完成文件的传输。根据客户端提交的服务请求（如上传、下载），存储模块执行相应的操作。

3）客户端

客户端与调度模块和存储模块不同，在一个独立的进程中提供服务，它只是以一个类库的模式存在，提供给用户上传、下载和查询等应用编程接口（Application Programming Interface，API）。当用户需要使用分布式文件系统的时候，只要提交服务指令和必要的文件信息，就可以使用分布式文件系统所提供的各种文件服务了。例如，当用户需要上传文件时，只需要提供需上传的文件名和本地路径。客户端首先和调度模块交互握手信息，表示自己需要上传文件，等待调度模块返回可用的存储服务器地址。在得到地址之后，客户端直接根据返回地址和存储模块交互，将文件上传到存储模块上。所有的交互信息都在后台实现，对于用户来说是透明的。

当用户登录文件系统时，即成为整个文件系统的客户端，通过向调度模块提交认证信息获得认证。如果该用户第一次登录，则在调度模块中设置文件保存该用户的认证信息，便于调度模块下次识别该用户。

8.3.4 基于虚拟内存的分布式文件系统

由于磁盘 I/O 性能的限制，分布式文件系统 CFS 的键值不适于海量小文件的存储，为了解决大规模数据存储的可扩展性，提出了云存储技术。

为了解决基于硬盘的存储系统所面临的各种问题，提出了基于随机访问存储器的云存储——RClouds。它将数据完全存储于 DRAM 中，其目标是提供优于基于硬盘存储 100～1000 倍的数据吞吐率和 100～1000 倍的访问延迟。因此从根本上对现有的数据存储模式进行了改进，包括重新设计交换机，重新设计副本策略、数据模型、数据分布策略、系统可扩展性方式、一致性维护方法等。

1. 体系结构

为了解决海量小文件存储系统读写速度慢的问题，本课题改进传统的多级存储结构，设计并实现了基于虚拟内存的分布式文件系统——RDFS。该文件系统具有高数据吞吐率、低延迟、高可靠的特性，设计的 RDFS 如图 8-6 所示。

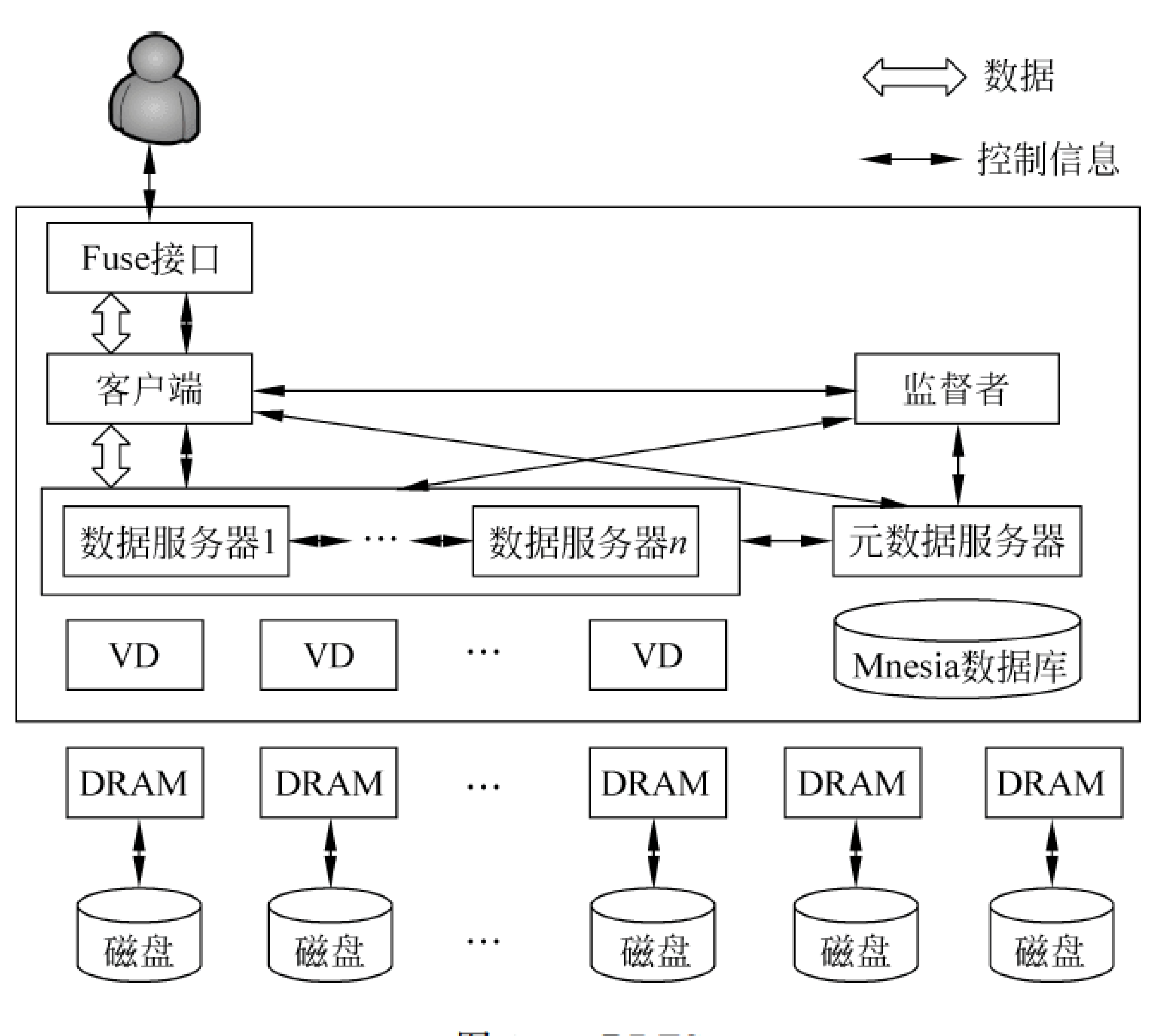


图 8-6 RDFS

RDFS 与传统分布式文件系统的区别主要有两个方面：将数据块存放在虚拟内存中，利用内存的随机读写优势来获得高效的数据读写性能；设计了监管者模块以保证系统的高可靠性和高可用性。RDFS 主要包括 VD、Mnesia 分布式数据库、元数据服务器、数据服务器、客户端和监管者六个模块。

VD 模块实现了基于虚拟内存的数据存储。将数据直接存储于物理内存的效率并不高。这是因为物理内存和内核存储器是分开维护的，直接将数据存储于物理内存中会造成两个不足：

(1) 浪费了没有被文件系统使用的物理内存空间；

(2) 数据会在内存盘和内核存储器中重复存储，维护时需要更新多个数据备份。

VD 将文件全部存储在 UNIX 内核管理的虚拟内存中，能够使用已有的内核接口、系统

资源和系统工具，能够根据数据量的大小动态地调整文件系统的占用空间，并能尽量减少对其他内存资源的影响。VD 能够大大减少磁盘和网络 I/O，性能接近于基于内存的文件系统。但是，它无法应用于分布式存储的环境，本文修改了 VD 的应用程序接口，使其符合 RDFS 其他模块的调用方式。Mnesia 是一个用 Erlang 编写的多用户的分布式 DBMS。本文使用 Mnesia 分布式数据库存储文件的元数据，实现了元数据的分布、可靠存储。

2．系统实现

1）元数据服务器

元数据服务器负责管理所有文件的元数据信息，主要包括命名空间，访问控制信息，文件到数据块的映射关系，数据块的位置信息等。除此之外，元数据服务器也控制系统级别的活动，主要包括数据块的分配管理、数据器状态管理和数据服务器之间的数据块镜像管理等。元数据服务器的效率对整个系统性能具有重要的影响。为了实现元数据服务器的高效、可靠和可扩展，RDFS 的元数据服务器遵循三点设计原则：①尽量减少元数据服务器和其他结点的网络交互次数，充分发挥虚拟内存的速度优势；②采取被动响应模式，降低元数据服务器的负载；③使用 Mnesia 分布式数据库实现全局命名空间的层次化、持久化、可扩展和自动故障恢复。集中式元数据管理被广泛采用，并获得了良好的性能。近年来，由于集中式元数据管理在可扩展性方面的局限性，越来越多的研究针对分布式元数据管理。RDFS 充分利用 Mnesia 分布式数据库的特性，实现元数据的高性能、高可靠存储。

2）数据服务器

在 RDFS 中，每一个文件拆分成固定大小的数据块，数据块大小可以根据不同的需求设置。创建文件时，系统会根据文件创建时间采用标准的 UUID 算法为每一个数据块生成一个全局唯一的、不可更改的 128b 数据块 ID。数据块 ID 由客户端生成，而不由元数据服务器分配，这样能够减少元数据服务器的负载和网络交互次数，使系统的整体性能得到提升。

3）客户端

RDFS 的客户端是应用程序访问文件系统的入口，它提供了包括 open、read、write、close、list、delete 和 mkdir 等访问系统的应用程序接口。客户端向元数据服务器请求元数据之后，直接向数据服务器请求数据块，以减轻元数据服务器的负载。

4）Fuse 接口

除了提供传统的 API 接口，用户可以使用 Fuse 开发库将远程目录映射到用户的本地目录。映射完成后，用户可以像操作本地文件一样访问 RDFS 中的对象，传统的应用无须任何改动便可透明地使用 RDFS 提供的存储服务。本文为 Fuse 接口设计了 128KB 的写缓存，8～512KB 的自适应读缓存，它能够显著提高小文件的读写性能。

5）监管者

RDFS 将进程清晰地划分为工作进程与监控进程两大类，工作进程负责执行正常的事务，监控进程负责监控工作进程状态和故障恢复。监管者的工作方式是一对一的进程监控，即一个监控进程监控一个工作进程。RDFS 采用一对一的进程监控为每一个元数据服务器、数据服务器和客户端工作进程启动一个监控进程。若工作进程失效，则监控进程会重新启动它，实现进程的不间断运行和自动恢复。对于多次重启仍然失效的进程，监管者会中断该进程并更新相应的元数据和数据。

3. 可靠性分析

1）元数据可靠性

元数据可靠性保证采用热备的方式实现，即将一张数据库表复制到多个结点上。数据库表的所有副本是等同的，并都存放在虚拟内存中，以实现元数据的快速读写。若对某张表进行修改，则每个事务中的写操作会在所有的副本上执行。这些操作对应用程序来说都是透明的。另外，监管者会启动一个监控进程来捕获元数据服务器的运行状态。当元数据服务器失效时，该监控进程会立即接管元数据服务器的任务继续执行。

2）数据备份策略

RDFS 利用数据备份来解决停电或成片务器掉电的问题。在双电源市电都停运时，系统采用 UPS 供电，并启动系统备份程序将元数据和虚拟内存中的数据保存到磁盘。待所有数据保存完成之后，停止系统运行。RDFS 重启时，将元数据载入 Mnesia 数据库中，并将数据载入至虚拟缓存中。待所有工作完成后，系统即可提供服务。

3）数据副本策略

默认情况下，RDFS 为每一数据块保存三个数据副本，分别存放在本机、本机架内的另一台数据服务器和跨机架的数据服务器的虚拟缓存中。对于不同的数据可靠性要求，用户可以指定数据块的备份数量。通常，在创建数据块、重新复制数据块和数据块均衡时需要创建副本。如数据服务器失效，某一数据服务器硬盘故障，某一数据副本故障，或者增加了副本数量等，均会创建一个新副本。当数据块的副本数量小于用户指定的备份数量时，监管者会立刻重新复制一个数据块副本。

4）心跳机制

数据服务器每隔 5s 与元数据服务器进行一次心跳通信，向元数据服务器传递数据服务器的状态信息。元数据服务器超过指定时间没有收到某一数据服务器的心跳信息，即认为该数据服务器失效，并启动错误处理程序更新相应的元数据和数据副本。

8.4 本章小结

本章首先对云计算以及云计算的关键技术进行了概述，然后对云存储的系统架构和优势进行了分析，接下来对 CFS 设计并实现了一种适用于云计算条件下的分布式文件系统，应用了负载均衡技术，良好地适应了分布式系统的动态性和高时效性，实现了大容量数据的存储和同步等功能，并且具有良好的可扩展性。

参考文献

[1] 中国云计算网——云存储专题. http://www.cloudcomputing-china.cn/Article/ShowClass.asp?ClassID=7.

[2] 云存储架构详解. http://www.cloudcomputing-china.cn/Article/luilan/201003/564.html.

[3] 云存储技术及其应用. http://www.ccsa.org.cn/article_new/show_article.php?article_id=technic_e49da337-959d-0b43-baeb-4c5f6ef4577d.

[4] 云存储的九大要素. http://www.cloudcomputing-china.cn/Article/luilan/201002/400.html.

[5] The Hadoop Distributed File System：Architecture and Design. http://hadoop.apache.org/core/docs/

r0. 16. 4/hdfs_design. html.
[6] Hadoop HBase Performance Evaluation Introduction. http://www. cs. duke. edu/～kcd/hadoop/kcd-hadoop-report. pdf.
[7] Ghemawat S,Gobioff H,Leung ST. The google file system [J]. SIGOPS Operating Systems Review, 2003,37 (5):29243.
[8] Apache Hadoop. HDFS Architecture. [EB/OL]. [2009202209]. http: // hadoop. apache. org/ common/docs/ r0. 20. 1/ hdf s2design. html.
[9] Happy2fish. fastDFS [EB/OL]. [2009202218]. http: // linux. chinaunix. net/ bbs/ forum27521. html.
[10] TFS Project. http://tianwang. grids. cn/projects/tplatform,2008.
[11] Borthakur D. The Hadoop Distributed File System: Architecture and Design. http://hadoop. apache. org/core/docs/current/hdfs design. pdf.

CHAPTER 9

第9章 计算安全

9.1 对等网络安全问题

9.1.1 研究背景

随着现有P2P系统的广泛、深入的应用，P2P系统逐渐暴露出一些安全问题，其中一个主要原因是存在一些诸如结点自主行为引起的不可靠的服务质量、网络攻击等。P2P系统具有的匿名性、高度的开放性以及加入系统的用户结点类型、目的、利益空间差异性大等因素致使结点行为也不尽相同，按照结点自主行为的不同将这种情况分为以下两种：

(1) 自私行为。如搭便车(Free-Rider)和公用地悲剧(Common Land Tragedy)。其中Free-Rider指结点只使用其他结点提供的资源或服务，而不共享自己的资源。以Gnutella文件共享系统为例，高达70%的结点是Free-Riders[1]。公用地悲剧指网络资源作为一种非排他的公共资源，被大多数P2P结点无节制地使用。结点之间的不合作及利己的自私行为严重影响了P2P服务的可用性。

(2) 恶意行为。P2P网络中还存在着为数不少的蓄意提供不可靠的服务质量以及欺诈行为的结点。这些结点并不关心资源访问，仅为了散布无效的或有害的内容，如虚假的文件、病毒和木马等，它们对所有的查询都做出积极的响应(无论是请求下载文件还是请求推荐信息)。恶意结点的存在严重影响了P2P系统的性能，使得P2P系统的可用性得到极大的破坏。

本节通过激励机制、信任机制和文件安全机制对结点的行为进行约束，引导P2P结点更倾向于成为一个“好”行为的结点，构建一个良性安全的对等网络。

9.1.2 激励机制

P2P系统尽管具有系统容错性高、容量大、可扩展性好等优点，但是缺点

同样明显。由于 P2P 网络的分布式特性,单个结点具有更大的自由度,服务质量因此无法得到保证;结点动态性使得服务不稳定;结点的自私性导致只有少部分结点愿意提供服务;同时少部分恶意结点更容易对其他结点发起攻击,这些攻击包括针对文件的攻击和针对路由的攻击[2]。系统测量表明[3],P2P 网络中大量用户(约 70%)只获取服务,不提供服务,这导致了 P2P 网络较为严重的不公平性问题。还有一部分结点通过欺诈手段增大自身利益,针对文件进行攻击,例如随意伪造文件,随意中止服务等[4],甚至有的结点之间互相勾结联合进行攻击(Sybil Attack[5])。在传统网络中,由于服务总是由中心服务结点提供,公平性问题和服务的安全性可靠性问题并不突出,但是在 P2P 中,需要激励结点提供安全可靠的服务。

1. 搭便车问题

P2P 网络资源的丰富依赖于用户对自己可用资源的共享。但是,真正在网络上创造或提供内容的人还是少数的。据统计 Gnutella 的用户中仅有 2%向其他用户提供了内容,即使在比较活跃的 Usene 七张贴文章的用户也仅占所有用户的 7%。虽然 P2P 模式中有着传输速度大和共享性质的种种优点,但现实中往往出现人们只想得到别人的共享文件却没有把自己的有用资源共享出来的问题,由此产生了 P2P 网络应用中的一个广泛存在的消极现象——搭便车。搭便车定义为一个自私的个体有意识地拒绝为某个群体的共同利益自愿地贡献。Saroiu 等在 2002 年发表的论文[6]中指出,根据他们对 Napster 和 Gnutella 的实验的最新结果表明:大多数用户只作为消费者而不对系统贡献自己的资源,50%的“种子”在线时间不超过 1h,多数的用户都是搭便车者,对系统贡献很少。比如 Gnutella 系统中有 25%的用户根本不共享任何资源。“种子”的短缺意味着系统中部分有价值的资源长时间得不到利用,搭便车现象的增加使得整个系统失去 P2P 分布式共享资源的精神,这一现象的蔓延,将导致 P2P 退化成传统的 C/S 系统[7]。由此引发了许多关于 P2P 激励机制方面的理论研究课题和商业计划,提出 P2P 激励模型,向系统贡献资源的用户能从网络中得到相应的回报,激励用户共享资源以消除搭便车现象。

2. 现实情况

在 P2P 网络中,尤其是纯 P2P 模式的网络中,并没有一个中心结点或是权威对合作进行监督。每个结点都是理性的,它们追求的是自身效用的最大化。如果与其他结点合作,提供资源或服务,将会消耗结点的资源,并降低结点的性能。尽管理性的结点可以预见缺乏合作会导致 P2P 网络的总体性能的下降,但由于合作所带来的好处并不直接,因此结点合作的动力不大。有研究人员对一个具有 35 352 台主机的实际运行的 Gnutella 网络进行了 24 小时的监控,发现在 Gnutella 网络中搭便车现象盛行,近 70%用户是搭便车者,即它们未被共享任何文件;近 50%的回应来自仅 1%的用户。研究人员认为,P2P 网络需要自发的合作,但这对于匿名的大规模系统很难做到;理性用户关心效用,并会影响其是否选择合作。

一个缺乏合作的 P2P 网络中,搭便车现象非常普遍,很多结点总下载资源或索取服务,从来不提供资源或服务。更有甚者,有的结点提供假的资源或服务,这种现象在 eMule 网络中也屡见不鲜。有些结点故意限制或谎报网速,以达到少提供或不提供资源或服务的目的。所有的这种种行为,导致了公用地悲剧[8]:网络资源集中化,造成网络拥堵。另一方面也导致网络资源同质化,结点不愿意主动引入新的资源。最终使得整个 P2P 网络性能急剧

下降，所有参与其中的结点的利益都受到不同程度的损害。

上述问题的根源在于，目前的 P2P 网络，无论是采用何种架构或模式，都基于一个假设，那就是每个参与的结点都能善意地、最大化地提供网络资源。这个假设忽略了一个事实，那就是每个结点后面都是具体的人，结点的行为实际上就是个人的行为。因此，可以通过研究个人的行为来解决这个问题。作为组织行为学的重要组成部分的激励机制，在这里可以发挥巨大的作用。

3. 面临的问题

尽管传统的激励机制在实际应用中已经展现出它的巨大威力，然而 P2P 网络具有其特殊性，因而不能照搬传统的激励机制，而是应该从 P2P 的实际情况出发，制定出适用于 P2P 网络的激励机制。在制定之前，需要对 P2P 网络中激励机制可能面临的问题[9]进行研究。

传统的组织有大有小，但总的来说，一个组织的员工并不会太多。以全球最大的软件公司——微软为例，其业务已经遍及全球 90 多个国家和地区，其全球员工总数将近 60 000 人，它可以算是一个巨型的组织了。而根据统计[6]，在一个像 Gnutella 和 KaZaa[10] 这样的文件共享系统中，只是其并发的用户数完全可能超过 100 000。由此可见，不同于传统组织，P2P 网络可以拥有非常多的成员(结点)数量。

在传统的组织中，尽管也存在着员工离职或新员工入职，从而存在一定的周转率，一般来讲，这个周转率并不会太高(如果有组织有很高的周转率的话，工作效率的降低往往是因为员工交接及新员工需要时间适应新环境和新工作引起的，这并非激励机制所能解决的)。然而，根据研究人员的统计，在像 Gnutella 和 KaZaa 这样的文件共享系统中，结点从加入网络到离开网络的时间间隔，即“生存时间”，其平均值基本上是以分钟计算的。由此可见，P2P 网络具有很高的周转率。

传统的组织中，员工往往以部门或项目为单位进行组织管理。在同一个单位中的员工，往往具有相同或类似的兴趣，这为激励机制的应用带来了很大的方便。而在 P2P 网络中，由于结点都是对等的，因而并不存在这样的单位。而且，结点之间的兴趣也是不对等的，比如 A 可能对 B 的资源或服务感兴趣，而 B 却对 C 的资源或服务感兴趣。P2P 网络这种兴趣不对等的特征对于传统的激励机制来讲，是一个巨大的挑战。

在 P2P 网络中，多个结点可能“串通作案”，以提高自身的信誉。这种情况在一些采用了基于信誉的激励机制的 P2P 网络中非常普遍。在传统的组织中，尽管也存在类似的问题，但传统的激励机制中信誉不是主要的因素，因而这种行为对传统激励机制的影响比较小；另外，在 P2P 网络中的这种行为更难察觉，具有更大的危害性。

P2P 网络还带来了两个在传统的组织中不存在的问题。一个是零成本身份切换，在 P2P 网络中，由于结点都是匿名的，因而结点可以任意地更改自己的身份。另一个是行为背叛的问题，即一个在历史上行为良好的结点，突然更改其行为，变成一个搭便车者，或做出其他危害 P2P 网络的行为。

4. 激励模型

在 P2P 网络中，激励模型可以大体被分为两类：①货币支付模型，即结点间每次文件的上传和下载都需要明确的货币支付；②非货币模型(又称软激励模型)，即不使用明确的货币支付而采用其他的方法提供激励。

1）货币支付模型

（1）真实货币支付。用户在网络中出售自己的资源，利用了用户资源的网格组织通过银行向该用户支付真实货币，付费在网络外进行。这种方式广泛地应用在网格计算的经济模型中，目前的一些网格系统如 Globus、Legion 等已经提供了大量的、成熟的、可重用的中间件，例如资源协同分配服务 DUROC、认证和安全服务 GSI 等。

（2）MicroPayment。在这种模型下，所使用的货币并不能兑换为现实货币，可以称为虚拟货币。每个用户下载资源之前必须向服务提供结点支付相应价格的虚拟货币。为网络中其他结点提供服务可以得到相应的虚拟货币，所以为了能够持续地得到网络资源，结点必须不断地以自己的服务换回足够多的虚拟货币才行。其中货币的交易必须具备 ACID，即 Atomicity（原子性）、Consistency（连贯性）、Isolation（孤立性）、Durability（耐久性）。现实中的系统包括 MojoNation[11]、Maze 等。

2）软激励模型

（1）实物交换模型。用户总是只有共享了文件，对系统做出了贡献才能从别人那里下载文件。上传与下载的速率同样受到控制，即用户使用一定的速率下载文件，也必须要提供相应的上传速率。使用这样的机制的系统目前有 eDonkey 和 BitTorrent。

（2）区分服务质量模型。向系统贡献越多，则可以得到更好的服务，比如优先的访问权，更为稳定的传输，更高速的下载，更大的存储空间等。例如在参考文献[12]中所提出的根据结点服务情况和使用情况矩阵特征向量来决定是否向请求服务提供结点提供其所要求的服务。

对于如何在 P2P 上建立激励模型，采用什么样的激励机制及算法，众多研究者进行了大量的研究和实验。目前在 P2P 网络中引入激励机制的主要方法有：①重新设计分布式的资源定位协议来实现网络激励，如对于网络贡献大的结点可以把资源搜索报文传输到更多的结点，进而增大结点资源的发现概率。②在已有的协议上增加特殊的激励算法，如以请求的接纳控制、服务质量的区分等来实现网络激励。

现在大量的研究工作集中在在现有路由协议上增加特殊激励算法来实现激励。这主要是因为现有的资源定位协议，不论是使用无结构泛洪还是使用有结构分布式哈希函数来定位资源的协议都基本成熟，并且有大量的应用，抛弃现有的大量系统而重新开发基于新协议的应用代价过大。采用在现有协议上增加激励的方法，不仅理论上更容易实现，而且解决了现实中大量存在系统的缺陷，有更加现实的意义。

5. 激励机制研究现状

1）基于微支付和虚拟货币的机制

斯坦福大学的 Phillipe Golle 等人首先提出了使用微支付和虚拟货币的方法来解决 P2P 共享网络中的共享激励问题[13]。这个机制主要思想是，服务器记录每个注册用户文件下载数量 u 和文件上传数量 v，每次用户间传递文件时，服务器将提供文件下载的用户的文件上传数量 v 加 1，并将下载文件的用户的文件下载数量 u 加 1。每隔一段时间，服务器为每个用户计算应支付的金额 $C=f(u,v)$。这里的函数 f 根据一定的算法将用户的下载与上传差额换算成用户应该支付的金额。函数 f 一般采用线性函数，以便使得整个网络中的微支付总额为 0（收支平衡）。

为了增加该机制的灵活性，又引入称为“点数”的虚拟货币。服务器不再记录用户的文

件下载数量和上传数量，而是记录用户的点数。每次用户间传递文件时，服务器增加提供文件下载的用户的点数，同时减少下载文件的用户的点数。用户可以通过提供文件下载来获得点数，也可以直接使用现实货币购买点数。在使用现实货币购买点数的情况下，由于交易额一般很小，直接通过银行支付费用相对较高，而且比较麻烦，因此需要有第三方来向用户销售点数，再由其与银行结算。这个第三方称为经纪人(Broker)。

这样一来，经纪人结点的稳定性在这种机制中就至关重要了。因此，PPay[14]引入并实现了浮动的、自管理的货币的概念，最大限度地减少经纪人的介入。浮动的货币允许数字货币(即虚拟货币)在没有经纪人参与的情况下，也能从一个结点“浮”到另一个结点。至于说自管理，是指在 PPay 中，由拥有数字货币的结点自身来负责数字货币的安全，只有在购买数字硬币或兑换成现实货币时才需要经纪人的介入。

Fileteller[15]是一个网络文件存储系统，它也采用微支付方式来对网络中的用户进行激励。

在支付系统中，资源或服务以商品的形式存在，消费者要购买使用权，而提供者可以获得报酬。P2P 系统中，结点通过向其他结点提供资源或服务来获得报酬，并用获得的报酬去购买自己所需的资源或服务。由于 P2P 系统中的资源常常是比较廉价的(例如空闲的带宽或 CPU 时间等)，所以在 P2P 中采用的多是微支付(Micropayment)系统，每笔交易的价值较小，对安全性等方面的要求也相对较低。

学术界研究文献中提出的比较典型的 P2P 支付系统有 PPay[14]、KARMA[16]、PeerMint[17]等。实用中的 P2P 微支付系统实例有 Popular Power 和 MojoNation[11]，前者付少量的报酬购买别人的空闲 CPU 时间用以构造 P2P 分布计算网络，再出售给需要的用户，后者是一个 P2P 在线支付系统。

2）基于配额的机制

基于配额的机制的主要思想是，为每个结点设定一个配额，结点在一个时间段内从 P2P 网络中的下载总量不得超过这个配额。

CFS[18]是一个只读的 P2P 网络存储系统，为了减少攻击，提高系统的安全性，它采用了存储配额机制，规定每个结点只能访问存储系统总量的一个很小的比例，例如 0.1%。

CFS 引入配额的动机并非是为了激励，而是为了提高系统的稳定性和安全性，而 FARSITE[19]和 Pastiche[20]则将配额用于激励机制。这两种激励机制都是通过限制结点能够从 P2P 网络中获取的资源与其对 P2P 网络的贡献相当，从而激励用户贡献资源。

Samsara[21]则采用了一种比较特别的策略：每个结点在向其他结点请求存储空间之前，必须允诺对方能够使用本结点上相同大小的空间，对于那些不遵守规则的结点，Samsara 会以一定的概率进行惩罚。

3）基于信誉的机制

信任是指一个结点基于个体体验，对另一个结点在系统中可信度方面的一个评价，而信誉则是指一个结点通过合作的方式，基于自己或者其他结点的一些信息来获得其他结点在系统中的可信度方面的一个评价。

P2P 系统中信任和信誉关系的基本思想是用户间完成交易后，可以对这次交易进行评价，从而给对方一个评价。用户间可以通过这些相互间直接的评价来建立对对方直接的信任关系。同时，这种直接的信任关系可以通过某种信任传播算法来描述用户在系统中的主

观或者客观的信誉值。

信誉模型主要分为主观的信誉模型和客观的信誉模型。主观的信誉模型是指每个结点都建立并维护一个信誉表,这个表中存储了结点对每个有过交往的结点的信誉的评价。而客观的信誉模型中用户不用单独维护信誉表,而是在整个 P2P 网络中,所有结点以某种机制共同维护一个全局信誉表,这个表中存储了所有结点的全局信誉值。

SLIC[22] 是一个典型的主观的信誉模型。它的基本思想是:每个结点统计各个邻居结点对本结点发出的请求的响应情况,然后根据响应情况给其打分,响应越好,得分越高。这个打分每隔一段时间进行一次,打分的结果作为接下来的时间段中本结点对邻居结点的请求进行响应的依据。每个结点既要响应邻居结点的请求,又要发出自己的请求,而且每个结点的能力是有限的,因此它必须要在响应请求和发送自己的请求之间找到一个平衡点,使得在为邻居结点服务的同时不影响自己从 P2P 得到服务的质量。

主观的信誉模型实现比较简单,只要发生交互的两个结点的参与,但是,对于那些为邻居结点提供过良好服务的结点,当它们与新的结点交互时,其良好的历史记录并不能为其带来好处,这样对它们来讲并不公平,一定程度上会降低其积极性,而且容易造成结点的功利性:需要时便向邻居结点提供服务以得到好的服务,不需要时便拒绝提供服务。

客观的信誉模型实现起来要比主管的信誉模型复杂,而且容易受到攻击和欺骗。然而,相比于主观的信誉模型,它帮助形成了更公平的环境,结点即使当前不需要获得服务,它也可以提供良好的服务,提高自己的信誉值,以便将来或遇到新结点时能够得到更好的服务。

4) 基于 TFT 的机制

Tit-for-Tat(TFT)是一种非常简单的合作策略:第一次交易中总是选择合作,之后每次交易采用的策略与对方在上次交易中所采用的策略(合作或欺骗)相同。正如 R. Axelrod 在《The Evolution of Cooperation》一文[23]中所指出的那样,TFT 是自我主义者构成的无中心交易环境中最好的合作策略。目前最受欢迎的 P2P 文件共享和内容分发系统——BitTorrent(BT),就使用了这种动机机制,并在改善公平性方面取得了良好的效果[24]。

在 BT 中,TFT 策略通过"阻塞(Choking)"算法来实现。在一个 Torrent(同一个文件的下载群)中,每个结点周期性地计算其所有连接上的下载速度,并根据这些信息,选择其中速度最快的 w(默认为 7)个连接提供上载,其他连接除了一个由"乐观疏通(Optimistic Unchoking)"选定的之外,全部予以阻塞(choke)。乐观疏通的目的是为了有机会找到一个更好的连接。如果该连接上的下载速度优于某个目前正在提供上传的连接,则这个新连接接替其在 w 个连接中的位置,否则,下一轮将以 Round-Robin 方式选择另一个乐观疏通的连接[24]。

BT 的激励机制可以使尽量多的有效连接处于双向传输的最佳利用状态,并且,为了获得更快的下载速度,每个结点也会有足够的动机在下载的同时也为其他结点提供上传服务,从而有效地减少了搭便车现象的发生。

5) 基于相似性的机制

2001 年,R. L. Riolo 发表在《Nature》上的一篇论文[25]提出了另一种合作机制。该文认为除了血缘关系和互惠关系(直接或间接)之外,合作还可能会在"相似"的实体之间产生。实体的特征可以用"Tags"来表示,这些 Tags 用来进行相似性的度量。参考文献[26]中提出了一种将 Tags 模型应用于 P2P 合作问题的框架。

9.1.3 信任机制

P2P 网络是一种典型的“开放式”网络环境。系统中的用户并非来自同一个利益团体，任何人只要愿意都可以自由地加入和退出 P2P 网络，自由地参与资源共享和交换。参与结点的身份对等，所有结点既可以是资源或服务的消费者，也可以是提供者。用户具有很强的自主性，因此提供的服务质量不可能像传统的 C/S 或 B/S 模式那样可靠，它们可以随意中止服务，甚至可能存在欺诈行为。另外，P2P 中的交互模式是点对点的，个体间的交互和协作是系统的主要业务，个体行为和结点之间的对等交互不被第三方直接介入或监控。

P2P 模式的开放性、对等性、自主性和无监督性是它的主要特征，也是它在很多领域取得巨大成功的重要原因。但这种个人为公众提供资源同时又享受公共资源而结点行为无约束的工作模式，使 P2P 网络中“信任”极度缺乏，交互双方很难判断对方的可信程度，交互对象的选择具有很大的盲目性，服务质量和安全也很难得到保证。

P2P 网络中的大量不可靠服务以及欺诈行为都是由于信任缺失而引起的。例如在众多文件共享 P2P 网络中，大量的文件是伪造的、未经授权的，或被篡改过的，甚至是带有病毒的。而在类似于 eBay 和淘宝网这样的电子商务类 P2P 系统（广义）中，这种不可靠服务和欺诈行为给用户带来的影响则更为严重。

1. 信任概念

信任是一个多学科的概念，描述了在特定的情境下，一个个体（A）在可能产生不利后果的情况下（包括风险因素），愿意相信另一个个体（B）具有某种能力或能够完成某项任务的主观信念[27]，或该个体（A）根据自己的经验或同时参考其他个体（C、D 等）推荐信息而得出的被信任方（B）的可信赖程度。Luhmann 于 1979 年从社会学的角度来描述信任，将其定义为减少社会复杂性的方法[28]。而这种复杂性是由具有不同理解力和目的的个体的交互引起的。该定义由于其社会学的本质更适合基于信誉的系统。另一个广为接受的定义是计算机科学家 Gambeta 于 1990 年给出的[29]，他将信任定义为个体评估另一个体或集体将执行某一特定行为的特定主观可能性等级，评估发生在个体能够观察到该特定行为之前（或该特定行为独立于个体能够观察到该行为的能力）且该特定行为会影响评估者自身的行为。

Gambeta 认为信任不是一个门限点，而应该是一个概率分布的概念，可以用介于完全不信任（用 0 表示）和完全信任（用 1 表示）之间的值来表示，且以不确定性为中点（用 0.5 表示）。该定义引入了从信任方的角度认识到的被信任方的可靠性概念。最近的关于信任的概念是由 Grandison 和 Sloman 提出的[30]，他们将信任定义为对某一个体在特定的情境下，独立、安全且可靠地完成任务的能力的坚固信念。Chen R[31] 认为：信任是大多数人际关系的核心，信任的因素因人而异，每一个人都有他自己的意见，因此信任的本质是分布的。

与信任紧密联系的概念是信誉，Abdul-Rehman[32] 将信誉定义为基于观察到的个体过去行为或过去行为的信息而对个体行为的期望。可以看出，信誉强调的是一个集体对某一个体（或群体）的综合的可信赖度，而信任更多强调的是信任个体对被信任方的主观信赖。

2. 信任模型

信任模型（Trust Model）是指建立和管理信任关系的框架。信任模型分为两种基本类型：层次信任模型、网状信任模型。层次信任模型是较为简单，并广泛使用的信任模型（如

X.509),其优点是结构简单,易于管理和实现,缺点是信任关系必须通过根来实现,层次模型适合孤立的、层状的封闭环境。网状信任模型中,每一个结点都可以作为可信任根,结点间的信任路径可以构成一个网络,如 PGP[33],网状信任模型的优点是更接近于人类社会的信任关系,信任关系易于构建,且不依赖于任何权威中心。

3. 信任模型的研究现状

信任模型是建立和管理信任关系的框架。目前,P2P 信任研究主要涉及信任量化、信任评价以及信任的计算、存储、传递机制的研究。相对于传统的安全技术,信任模型更像是一个不十分"严格"的安全技术。它不像一般的鉴别、认证等技术,有一个明确的接受或者拒绝的标准,而是更具有主观性。跟传统的安全技术相比,它建立了一个类似人类社会的信任评价和信誉传递机制,能更好地处理安全中的信任问题,一旦和已有的安全技术结合起来,就能对解决对等网中的安全问题提供较好的解决方案。

在分布式系统中,建立不同网络结点间的信任关系是建立系统安全的一个基础。P2P 信任机制主要是用来解决如何选择可信赖的 Peer 的问题的。信任问题在 C/S 时代也是存在的,但是在 C/S 时代构建信任机制要容易得多,因为 Server 处于中心位置,可以方便地收集 Client 的各项信息。但是由于对等网络的特点,信任模型是 P2P 网络应用中一个十分难以解决的问题。到目前为止,P2P 信任机制已经成为一个活跃的研究课题,尽管目前在实际应用中还没有出现比较成功的通用解决方案,但国内外研究者在信任研究领域开展了许多开创性的研究工作,一些具有代表性的研究成果提出了不少值得借鉴的思路和方法。

1) 集中式信任模型

集中式信任系统主要应用于电子商务领域,在实际应用中大都采用基于 PKI 的信任模型,简单地采用给参与者评价打分的方法来描述信誉度,采用简单的数值计算方式来实现信任的聚合。

在集中信任系统中,存在少数中心实体负责收集网络参与实体的历史交易记录信息,然后再把所有实体的信誉评分的结果公布出来。在下次的实体交易前,请求实体即可以通过参考备选服务实体的最新信誉信息来加以选择。这样拥有良好信誉的服务实体会获得更多的提供服务机会和回报,与此同时诚实可信的实体也会获得更高的信誉,从而抑制网络中的不良行为,最终会促进网络的良性发展。

集中式信任系统中,实体 A 和 B 在历史交易记录的基础上,在信任系统的支撑下相互选择对方,因为双方均认为对方的行为最可靠,从而开始一次新的交易交互。在每次交易结束后,实体之间会对对方在交易中的行为给出评价,信誉中心会不断地收集每个实体的评价信息并更新每个实体的信誉信息,更新后的实体信誉信息会在信誉中心公布。

在这类系统中,中心实体负责整个网络的监督,定期通告违规的实体,中心实体的合法性通过 CA 颁发的证书加以保证。这类系统往往是中心依赖的,具有可扩展性、单点失效等问题。这类实际系统的实例有 eBay、eDonkey 等。

2) 基于局部推荐的分布式信任模型

在这类系统中,结点通过询问有限的其他结点以获取某个结点的信誉度。一般采取简单的局部广播的手段,其获取的结点信誉度是局部的。如 Comelli[1] 对 Gnutella 的改进建议就是采用这种方法。

在局部信任模型中,许多研究者认为信任是主观的,对同一实体的可信度,不同的观察

者可能会得出不同的判断。例如，在 P2P 环境中，不同的参与者可能会采用不同的方法来评价其他参与者的性能，这反映了不同用户对行为的不同理解[34]（信任的上下文相关性特征）。在基于推荐的局部信任模型中，参与者通过询问其他有限的参与者（推荐者）来获得某个参与者的信任度信息，从而形成自身的信任观点。目前基于局部推荐的分布式信任模型的主要研究工作有：

(1) 基于概率的局部信任模型。在信任研究领域，部分研究者认为借助于概率方法可以描述主观信任，提出了多种基于概率的信任模型。

参考文献[35]中采用 Bayesian 公式对实体的信任相关经验进行了建模，提出了对实体间信任度进行定量研究的 Beth 模型。除 Beth 模型外，参考文献[34]中还针对 P2P 文件共享系统，提出了一个基于 Bayesian 网络的信任管理模型。该模型认为信任来自于参与者的直接经验，信誉则基于其他参与者的推荐，信任都具有上下文相关性、多面性、动态性等特点。

(2) 基于主观逻辑的信任模型。Josang 模型中将行为的结果（成功或失败）作为经验，根据二项事件（Binary Event）后验概率服从 Beta 分布的思想，提出了基于主观逻辑（Subjective Logic）的信任度评估模型来解决信任的推导和综合计算的问题[36]。

(3) Abdul-Rahman 模型。与基于概率的局部信任模型不同，Abdul、Rahman 等人[37]认为，尽管从直观上看，信任度可表示为某种概率的度量，但问题在于概率值只有以定义明确的可重复实验为基础才有意义，因而不适于处理日常的实际经验。并且，基于概率的模型仅考虑了观察本身，没有考虑观察者。此外，概率本质上是传递的，而信任只具有弱传递性。Abdul-Rahman 模型将信任划分为四级，分别累计不同级别的交易经验，并以此为基础进行信任的评价和推理。Abdul-Rahman 模型的不足之处在于其信任的表示和推理方法比较复杂，缺乏直观意义。

(4) PeerTrust 模型。Xiong Li[4]给出了一个适用于 P2P 电子社区的局部信任模型，结点的可信度是对以往该结点向其他结点提供服务的水平的综合评价。模型考虑全面，引入了结点对交互的反馈、反馈的可信度、结点参与交互的次数、交互的属性和结点所在社区五个因素度量结点的可信程度。其信任值的计算公式为

$$T(u)=\alpha\times\sum_{i=1}^{I(u,v)}S(u,i)\times \mathrm{Cr}(p(u,i))\times \mathrm{TF}(u,i)+\beta\times \mathrm{CF}(u)$$

相关参数的定义为：$T(u)$是结点 u 的信任值；$\mathrm{Cr}(v)$是结点 v 的推荐意见的可信程度；$\mathrm{TF}(u,i)$是结点 u 第 i 次交互的属性；$\mathrm{CF}(u)$是结点 u 所在社区的属性；$I(u,v)$是结点 u 和 v 之间交互的总数；$p(u,i)$是结点 u 第 i 次交互的对象；$S(u,i)$是归一化的结点 u 的第 i 次交互后 $p(u,i)$对它的信任评分；α 是与 u 交互过的结点对 u 的综合评价的权重；β 是社区对评估的影响所占的权重。

该模型对实体得到的推荐信任进行统计和分类计算得到实体的信任度，模型认为需要识别欺骗行为和对欺骗者进行惩罚，却没有提出具体的方法和机制。

(5) 其他局部信任模型。除上述局部信任模型外，其他研究者还提出了基于轮询投票的信任模型。其中，Damiani 等人基于 P2P 文件共享协议 Gnutella 提出了 Damiani 模型[1,38]。该模型以 Gnutella 协议的资源搜索与响应消息 Query 和 QueryHit 为基础，提出了轮询协议 XRep。参与者在下载资源之前先请求其他实体对某目标实体进行投票，投票

的消息采用公钥技术实现签名与加密。在对投票结果进行聚集分析和 challenge/response 检查，排除可疑投票之后，根据其结果确定是否访问该目标实体。在实际与目标实体进行交易之后，还要更新目标实体及投票者的可信度。

3）基于全局推荐的分布式信任模型

为获取全局的实体可信度，该类模型通过实体间相互满意度的迭代，获取实体全局的信誉度。Stanford 的 EigenRep[39] 是目前已知的典型的全局信任模型，该信任模型在信任问题研究领域具有重要的指导意义，成为大部分研究工作的参考标准。EigenRep 的核心思想是：依据一个结点提供的成功交易的次数与失败交易的次数来计算该结点的信誉值。当结点 i 需要了解任意结点 k 的全局可信度时，首先从 k 的交易伙伴中(曾经与 k 发生过交易的结点 i)来获知结点 k 的可信度信息，然后根据这些交易伙伴自身的局部可信度(从 i 的主观判断角度来看)综合计算出 k 的全局可信度。计算公式如下：

$$T_k = \sum_j C_{ij} C_{jk}$$

式中，T_k 为结点 k 全局的可信度，对于任意结点 i、j，C_{ij} 为结点 i 对结点 j 的局部信任度。

C_{ij} 计算公式如下：

$$C_{ij} = (\mathrm{Sat}_{ij} - \mathrm{UnSat}_{ij}) / \sum_j (\mathrm{Sat}_{ij} - \mathrm{UnSat}_{ij})$$

其中，Sat_{ij} 和 UnSat_{ij} 分别为结点 i 对 j 在历史交易中积累的满意次数和不满意次数。

参考文献[40]分析了 EigenRep 模型存在的不足之处，如缺乏迭代收敛性保证、没有考虑惩罚因素和网络性能开销等，并提出了基于推荐的 P2P 环境下的 Trust 模型，进行了相关分析，给出了分布式计算协议。

4）基于组群的 P2P 信任模型

基于组群的 P2P 信任模型[41,42] 以信誉为基础，通过将结点组织成组群来实现 P2P 系统安全控制。在参考文献[41]中，通过计算一个双层信誉，以此为依据最终选择一个结点进行交易，从而提高网络交易的质量和安全性。在该模型中，主要是通过逻辑上的一个结点组来实现双层信誉的(结点所在组的信誉和该结点本身的信誉)。模型规定每个结点在同一时间至多只能属于一个组。通过该结点所在组的信誉和该结点本身的信誉，可以判定这个结点是善意的还是恶意的，是否可以信任。一个结点的信誉可以通过该结点所要执行动作善恶可能性的大小来评定，并且随着不同结点之间的交互动态调整。如果一个结点滥用系统中的资源或者有自私的行为，该结点将受到降低信誉的惩罚。同样，如果有结点试图通过系统的匿名性伪装自己来攻击其他用户或更改其他用户的路由信息，也将受到惩罚。当其再次试图寻求协作时，系统将会提示这些结点是恶意的。另一方面，系统也将会给可靠诚实的结点一个好的信誉。当一个结点拥有较高的信誉时，该结点被认为是可信任的，并且可以获得所在组的支持。一个结点的可靠性越高，希望与其交互的结点就会越多。模型采用了双层的信誉模式，每一个结点行为的好坏同时还会影响所在组的信誉，这样可以激励组内的所有成员相互监督。在这种“监督”的压力之下，每一个结点都必须尽量做到最好，因为只有这样组才会批准它们进行交互。如果一个成员的行为有损组的形象或者削弱了组的信誉，以后组内的其他成员将会拒绝与其合作，以此达到惩罚和警醒其他成员的作用。若一个结点长期有不好的行为，组内的成员将有权选举决定将其从组中除去。

每个结点需要交易时，总是率先考虑与自己同组的结点，其次是组信誉度高的结点组中

的结点。提供文件时，也总是优先为自己的同组结点提供服务，其次再考虑其他组成员。因此，游离在各个信誉组外的结点很难找到机会与其他结点交互，信誉值的提高也会很慢，被一个结点组接纳需要比较长的一段时间。但是如果恶意攻击其他结点或是更改其他结点的路由信息，信誉的下降幅度却会很快，而且还有可能被踢出组。这样就可以起到一定的警示作用，使结点珍惜与组内其他结点的良好关系。

9.1.4 文件安全机制

文件安全机制是P2P分布自组织安全体系中的重要结构之一。文件的真实性问题和文件污染问题，得到了特别的关注，成为P2P诸多问题中关注率最高的问题之一[43]。

在无中心的P2P系统中，文件的真实性得不到保证，充斥着大量的虚假文件，甚至是威胁安全的恶意病毒或木马，这将严重威胁终端实体，甚至影响整个网络的安全稳定运行。有研究表明，几乎所有的热门资源都有虚假文件，而且比想象的多得多。一些虚假文件的传播量甚至远大于真实文件，这不仅极大地浪费了网络带宽，给网络系统带来严重威胁，对于依靠大量用户支撑的P2P网络来说问题相当严重。

对于文件污染问题，是指在P2P文件共享系统中，恶意结点发布与指示主题不相符合的文件内容，并通过P2P文件共享进行传播。文件污染问题给P2P文件共享造成了很大的危害：首先，如果用户频繁遭遇污染文件，其感受到的可用性会急剧降低，甚至最终放弃使用该系统；其次，它为病毒、蠕虫等恶意程序的传播提供了便利，造成了网络安全上的隐患。

对P2P网络的实际测量数据表明[44]，现实存在的文件污染现象十分普遍，尤其是对于最近流行的内容。在FastTrack/KaZaA、eDonkey、Overnet等P2P系统中，有半数流行内容的拷贝是被污染的或是仿造的。

1. 文件真实性概述

对等网络文件真实性是指：在对等网络中对于某请求结点的文件查询消息，会有不确定数量的应答结点给予应答，确定哪些应答是满足条件的真实的文件，这就是确定对等网络文件的真实性。举例来说，如果一个结点发起一个“国富论”的查询，结果收到三个应答，这些应答中哪一个是真实的呢？应答之一可能恰好是亚当·斯密所著的《国富论》。另外的一个应答可能是修改了几个关键段落的亚当·斯密的《国富论》。第三个应答可能是某个网络写手对《国富论》的恶搞作品，该作品的名字也被命名为《国富论》。判断这些应答哪个是真实文件的过程就是文件真实性确定。

文件真实性是对等网络的一个安全要求，然而到目前为止，收到的关注并不是很多，Daswani等提出过对等网络文件真实性认证的公开问题，他们在参考文献[45]中列举了四种不同的判断文件真实性的标准。

第一个判断的准则是“最古老的文件是真实的”。这种定义认为最早提交到系统中的文档是真实副本。举例来说，如果亚当·斯密是《国富论》的作者并第一个向系统提交了这个文档，那么他的文档就被认为是“国富论”这一查询的真实文档。任何在此之后提交的名为“国富论”的文档都被认为是查询的非真实回应。利用时戳机制的方法可以帮助这种系统确立文件的真实性。然而可以看出来，这种机制往往太机械简单。

第二个判断的标准是以专家为基础确认文件真实性。在这种方式中，一份文件签名如果经过了专家或权威结点的真实性确认，就被认为是真实的。比如，结点A是一个可信中

心，他提供对签名的确认。当结点收到查询请求的回应后，他可以与这个可信中心 A 通信，用结点 A 中的记录来确认签名的真实性。其实这种方式是借鉴了 C/S 模式的集中管理原理，可信中心结点 A 实际上就是一个确认真实性的服务器。这种机制存在着相当大的局限，如果结点 A 暂时或永久失效、被攻击者入侵控制甚至 A 本身就是一个恶意结点，那么文件的真实性就根本得不到保障了。这是一个单点失效的问题。

第三个判断的标准是基于信誉确认文件真实性。在现实生活中，在一个领域有很多专家教授，但是他们对这个领域的造诣是有高低之分的，有的专家教授的观点更加权威可信，有的则次之。与在现实生活一样，一些专家结点可能比另一些专家结点更可信，可以在投票中加大可信专家选票的权重。这种权重要由一个完整的信誉机制来提供，这种机制要能保持、更新、传播信誉值。现在已经有一些信誉机制的研究，但至今仍没有一个成功地被商业界利用。

第四个判断文件真实性的准则是基于投票来确认文件真实性。为了解决前面第二个方案中信任中心结点 A 的单点失效问题，这种方案用投票的方法让许多专家对文件签名的真实性进行确认，只需一部分专家认同就认为文件是真实的。这就不再有单点失效的问题。

2. 文件真实性确认协议

Ernesto Damiani 等人设计了 XREP 协议[38]，是一种具有代表性意义的文件真实性确认协议，XREP 提出了结合文件信誉度（Reputation）和参与结点的信誉度来确定对等网络文件真实性的机制。在这种机制里，系统的每个参与结点拥有一个结点标识符 servent_id（一般是用该结点的公钥进行哈希计算得到的），每个文件拥有一个文件标识符 resource_id（一般就是把文件内容进行哈希计算得到的）。每个参与结点都有一个经验库（Experience Repository），记录对文件和其他通信结点的某些历史评价信息，如用二元组（resource_id，value）记录文件的信誉度，value 以某种方法对文件的评价值；用三元组（servent_id，num_plus，num_minus）记录结点的信誉度，其中，num_plus 是在结点上成功下载文件的次数，num_minus 是在结点上文件下载不成功的次数。对结点的投票可以根据不同的标准，比如说一个简单的方法是，某结点只对成功下载次数为 0（num_plus＝0）的其他结点给下面评价。整个搜索、投票和下载的过程具体可分为五个阶段：

1）搜索资源

发起者 I 以类 Gnutella 形式的泛洪方式向所有邻居结点发送查询消息 Query。形如 Query（search_string，min_speed），系统中的结点在收到查询消息后立刻查看本地是否有符合查询请求的文件内容，如果有，就按照查询消息的发送路径返回一个查询响应消息。形如 QueryHit（num_hit，IP，port，speed，Result，trailer，servent_id），包括响应者结点标识符、查询到的文件名和其他信息组成的结果集 Result、匹配查询请求的文件数量、网速和〈IP，port〉对。

2）选择资源和发起投票

根据上一步返回的查询响应结果，发起者 I 在收到的 QueryHit 中根据响应者的信息选择一个资源 r 和一定数量的资源 r 的提供者组成集合 $T=\{S_1, S_2, \cdots, S_n\}$，这种选择可以取决于请求者的个人喜好和同种资源提供者的人数。I 产生一对密钥对（PKpoll，SKpoll），然后将发起投票的消息 Poll（r，$\{S_1, S_2, \cdots, S_n\}$，PKpoll）以类 Gnutella 的方式泛洪出去，由此

发起投票。系统中的结点收到发起投票消息以后，检查它们的经验库，根据对文件和结点的记录产生投票，投票用 I 的公钥加密按照投票发起消息的发送路径返回给请求者。形如 PollReply({IP,port,votes},PKpoll))，用公钥加密有两个目的，其一是防止消息在传输过程中被人恶意窜改，其二是保证投票和投票者的机密性，不让攻击者发现投票者和票的关联。

3）统计票数和核实投票

根据上一阶段收集的 PollReply，发起者 I 先用私钥 Skpoll 解密发现被窜改过的票并且将其丢弃，再根据 IP 地址排除派系。然后在排除派系后的所有投票者中选择一个投票者子集 V'，再用〈IP,port〉向每一个投票者 v_j（v_j 在子集 V' 中）直接发起投票核实请求消息 TrueVote，要求 v_j 向 I 发送核心信息 TrueVoteReply(response)核实投票。经过核实，I 相信某些投票，丢弃那些没有回复和没有通过核实的投票。

4）选择资源提供者并检查其可用性

请求者根据从阶段 3)中确定的可信投票选择一个信誉度最高的资源结点作为资源提供者 S。然而在资源下载之前，必须要核实资源提供者 S 的身份，因为要防止其他结点利用该资源提供者 S 的 servent_ids 冒充 S 提供资源下载。这个阶段的过程是：发起者 I 发送确认请求 AreYou(server_ids,r)给资源提供者结点，要求该结点作出应答，收到确认请求的结点用自己的私钥 SKs 加密确认消息后，连同自己的公钥 PKs 一同发送给请求者。形如 AreYouReply([response]SKs,PKs)，请求者收到 AreYouReply 后用 PKs 解密[response]SKs，得到确认结果，然后把 PKs 作哈希计算。如果哈希计算的结果是 server_ids，则证明发起者 I 是在和真正的资源提供者 S 通信。

5）下载资源

发起者 I 直接与资源提供者 S 通信，download(r)要求下载资源。下载后请求者会计算文件内容的摘要以确定完整性。最后发起者 I 更新经验库中资源和资源提供者的记录。

由以上对五个阶段的分析可知，整个机制可以防止攻击者修改投票，能够发现和排除派系选票，还能有效地挫败攻击者假冒资源提供者。像 Gnutella 网络一样，系统具有幂规律(Power Law)特性，即热门一些的文件会比不热门的文件更频繁地被搜索到，对它们的投票也会多一些。而且少数结点有较高的度，多数结点的度较低，因此少数结点将提供多数的资源下载。

3. P2P 文件污染概述

所谓文件污染是指 P2P 文件共享网络中的恶意用户，可称之为“污染者”，将虚假甚至含有恶意内容的文件贴上某些热门内容的标签进行发布，诱骗其他用户下载，并利用 P2P 网络的自由共享功能进行更广泛散播的现象。

P2P 文件污染的危害主要有三方面：一是降低了网络内共享资源的可用性；二是破坏了安全和互利的共享资源环境；三是为病毒、蠕虫的传播提供了便利。由于现有的 P2P 文件共享网络普遍缺乏准入控制和内容管理机制，所以，污染者可以像正常用户一样自由地发布和共享任何内容。再加上 P2P 网络中文件传播往往是“一传十、十传百”，而用户的行为又具有很强的自主性，很难加以管理，因此，文件污染一旦发生，将很难得到有效的控制。

P2P 文件污染最初是版权组织为了破坏版权文件在 P2P 网络上的非法传播而采取的一种比较消极的技术手段。从 2002 年起，一些公司便开始雇佣 P2P 污染者，在各 P2P 网上

布满其试图保护的音乐、电影、软件的假冒伪劣版本，欲使这些网络瘫痪。例如当时最著名的专业 P2P 污染公司 Overpeer，就曾经于 2003 年成功地使当时最受欢迎的 Kazaa/FastTrack 网络上被污染的文件占到总文件数量的一半以上。虽然从这些年的数据来看，这一技术并未真正达到版权保护的作用，但它对 P2P 文件共享系统造成的影响却吹响了 P2P 系统中内容安全对抗战的号角，并必将引发更大范围内不同目的性的文件污染和对抗以及更多的内容安全问题。因此，在现阶段分析讨论和防范文件污染不仅是个有关版权的议题，更是应对未来 P2P 系统中将要不断涌现的内容安全问题的一种必要的准备。

4. 文件污染方式

文件污染一般是针对某些选定的关键词进行的。文件污染者有如下三种污染方式可以选择[46]。

1）索引污染

最简单的文件污染方式是“索引污染”，即在 P2P 网络的索引服务系统中注入大量虚假的记录，这些记录指向不存在的版本/副本。当用户按照这些记录的指示尝试下载时，将得到“无法连接”的提示。如果注入的虚假索引记录足够多，那么没有耐心的用户可能在几次失败的尝试之后放弃下载的努力。

索引污染既可以针对版本也可以针对副本。它与普通的版本污染和副本污染的不同之处在于，污染者注入网络中的索引记录指向并不存在的对象，因此污染者并不需要拥有强大的污染服务器来提供大量的上传服务。

2）副本污染

所谓副本污染，指的是污染者声称自己存有某个正确版本的副本，但实际上传给下载者的却是错误的数据。如果这种污染者足够多，那么，即使下载者能够挑选出正确的版本，一旦误选这些污染者作为下载源，也会浪费大量的时间精力和网络带宽。

这种污染方式要求污染者拥有强大的污染服务器来提供大量的上传服务。

3）版本污染

最复杂也是危害性最大的文件污染方式是版本污染。实施版本污染的污染者首先针对一个（或同时针对多个）目标关键词制造出大量含有恶意或错误内容的污染版本。然后污染者将这些版本的索引信息注入目标 P2P 网络，并在其污染服务器上提供大量可供下载的副本。如果没有有效的识别措施和管理机制，网络中的用户在搜索相关主题时就很容易被这些具有大量可下载副本的污染版本所吸引。一旦下载了污染版本而又没有及时加以检验，一般用户很可能将该版本的本地副本设置为共享，并提供给其他用户下载。如此一来，污染版本将在网络中广泛的传播开来，甚至会超过了正确版本的副本数量，最终将正确副本淹没在污染副本中，使得该主题资源变得不可用。

P2P 共享文件的污染版本有很多不同的表现形式，例如，对于 MP3 歌曲文件，污染者可以采用截短、插入噪声、插入不可解码的数据片断甚至插入辱骂词句等方式来制造污染版本，而对于可执行文件，则可能是插入蠕虫、木马等恶意代码。由于 P2P 网络中共享资源的多样性，对文件版本的好坏，很难有有效的自动识别措施。因此，版本污染具有很强的隐蔽性，大多数情况下只能依靠人工的识别。正是这种人工识别的滞后性，使得 P2P 网络中被污染的文件版本不仅可以通过污染服务器直接散发，还可以通过正常用户的共享行为得到更加广泛和迅速的传播。

以上三种污染方式可能单独使用,也可能结合起来形成更为复杂和隐蔽的复合式污染方式。需要指出的是,索引污染只会降低资源可用性而不会引入有害内容,副本污染虽然可能引入有害内容,但很容易通过校验码等简单的机制加以识别(目前的 P2P 文件共享网络大都采用了这种机制),因此这两种污染方式单独使用时危害性相对较小。对 P2P 共享社区的内容安全威胁最大的是版本污染,这种污染方式需要特别的关注。以下除非特别指出,提到"文件污染"的地方都是特指版本污染。

由于现有的 P2P 文件共享网络普遍缺乏准入控制和内容管理机制,所以,污染者可以像正常用户一样自由地发布和共享任何内容。再加上 P2P 网络中文件传播往往是"一传十、十传百",而用户的行为又具有很强的自主性,很难加以管理,因此,文件污染一旦发生,一般很难得到有效的控制。最终的后果是造成 P2P 网络中充斥大量不可用的甚至是有害的索引信息、文件版本或文件副本,从而引起有害内容的扩散和资源可用性的降低,并最终造成用户的流失。

5. 文件安全研究现状

P2P 文件污染的报道最早出现于 2002 年,美国的 Overpeer 公司曾经成功地使当时最受欢迎的 Kazaa 网络上被污染的文件占到总文件数量的一半以上[47]。学术界对这一现象的研究从 2005 年开始。参考文献[44]中最先对 P2P 文件污染进行了正式的描述,随后,文件污染问题开始引起更多研究者的关注。根据关注角度和研究方法的不同,目前学术界对 P2P 文件污染的研究工作大致可以分为以下三类。

第一类是对现有 P2P 网络中的文件污染现状的测量分析。例如:参考文献[44]对 Kazaa 网络中的文件污染进行了详细的测量、统计和分析;参考文献[48]侧重于 eDonkey 网络;参考文献[49]侧重于 BitTorrent 网络;参考文献[43]同时对 eDonkey、KaZaa、Gnutella 和 OverNet 这几种流行的 P2P 网络上的内容可用性和污染问题进行了广泛深入的测量和分析。这些测量分析很好地揭示了 P2P 网络中污染文件的静态空间分布,但大多并不能揭示出污染扩散过程的动态时间演化规律。

第二类是对 P2P 环境中文件污染问题的抽象和建模研究。参考文献[50]中的 P2P 文件传播模型借鉴了疾病传播的 K-M(Kermack-Mckendrick)模型。作者用他们的模型分析了单个版本文件的传播规律,并推导出保证传播成功的"离开率"门限。他们还通过仿真实验分析了两个版本(一个正常,一个被污染)的竞争传播。参考文献[51]采用了类似的方法为 P2P 病毒和文件污染散播建模,还考察了结点动态进出系统的影响,以及信誉系统的作用。参考文献[52]中的模型较为简略,基本上遵循了相同的思路。但该文中对用户行为的问卷分析结果提供了非常有参考价值的两个结论,即同一用户对不同类型的污染所具有的警觉性可能是不同的,而不同用户对同样的污染也可能具有不同的警觉性。

第三类研究工作的主要内容是探讨如何防治 P2P 文件污染或者对污染内容的扩散进行控制。由于 P2P 文件污染现象出现的历史短、规模大、情况复杂多变,所以这一类研究工作大多是尝试和探讨,并没有得到广泛公认的成功先例。讨论较多的一种方案是针对共享文件的内容可用性而建立的"对象信誉系统",关于这种方案的细节可以参见参考文献[53]。

9.2　云计算平台相关技术

9.2.1　研究背景

随着计算机软、硬件技术的高速发展，新的计算模式也相继出现，继分布式计算、并行计算、网格计算、效用计算等概念的发展，又出现了一种新的计算模式——云计算[54]。云计算的演进过程如图 9-1 所示。云计算是将网络中的计算资源整合起来形成超大规模的资源池，以服务的形式按需提供给用户，它既是一个学术概念又是一个商业概念[55]。

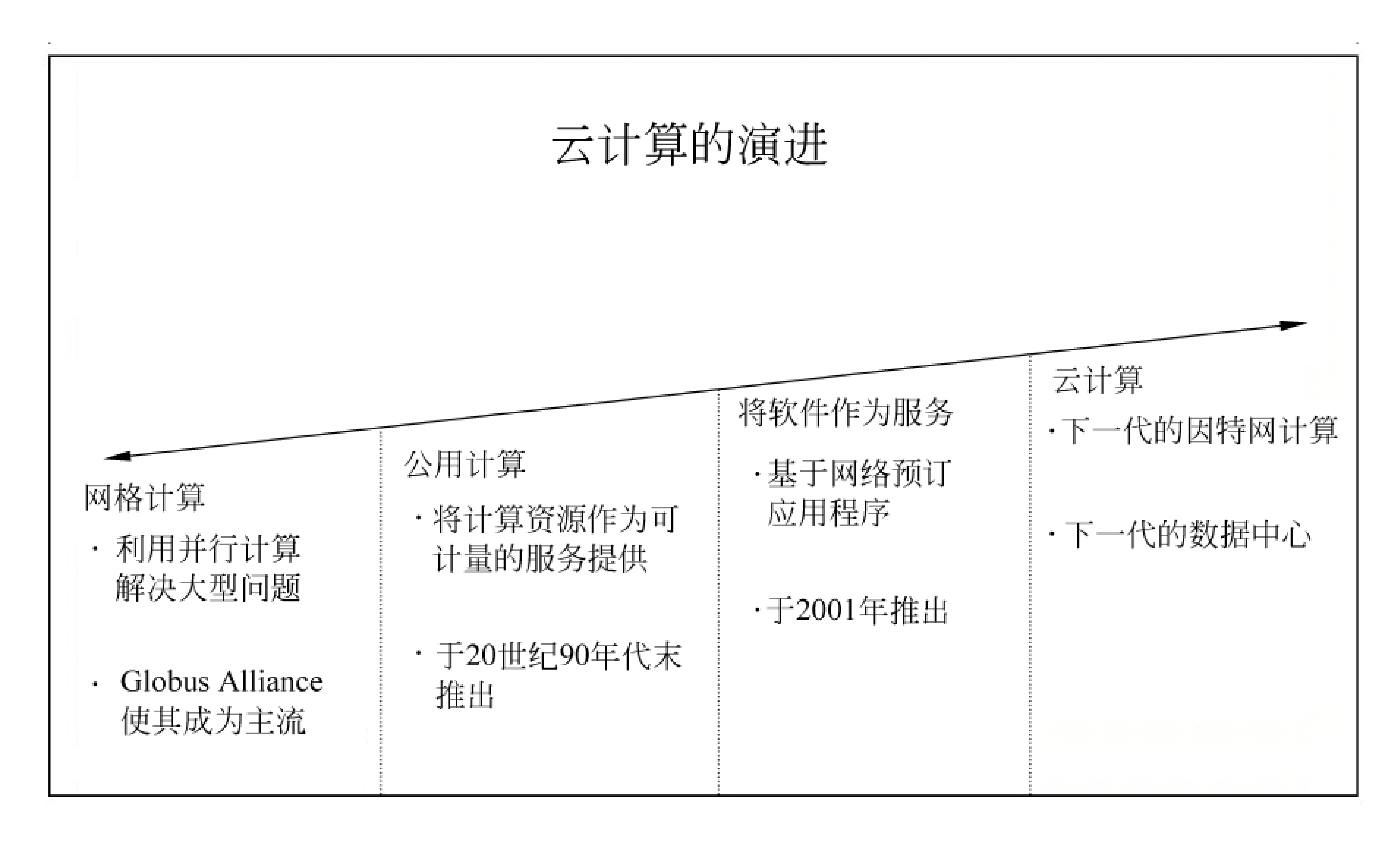


图 9-1　云计算的演进

云计算的概念自 2007 年被提出以后，就成为国内外计算机网络技术研究的一个热点。目前已有 IBM、Google、Amazon 及微软等业界的一些巨头公司开始提供云计算服务了。云计算几乎成为所有 IT 业内巨头的主要发展战略之一[55~57]。

云计算将计算推到了云中，也将人们的日常生活与“云”紧密联系在一起。因为云计算从一开始定位的服务对象就是所有普通用户，可以是个人，也可以是商业性质的企业或组织[58,59]。这就意味着云计算具有无限商业潜力，所以云计算的概念一经提出，马上获得了众多 IT 企业(尤其是一线 IT 企业)的高度关注，并迅速发展成为转变传统计算模式的一种革新技术。Amazon 公司的弹性计算云(Amazon Elastic Compute Cloud，EC2)与简单存储服务(Simple Storage Service，S3)成为云计算的一个成功案例[60,61]。

云计算是对传统计算模式的一次革新，同时也是对计算机的服务商业模式进行的一次改革[61]。在传统的计算模式中，用户需要自己管理复杂的计算机硬件和软件资源。而通过云计算，用户只需要关注自己需要什么类型的服务。将应用服务与计算资源分开可以很大程度上降低用户信息化的复杂度。云计算的资源整合大大提高了计算资源的利用，同时能够屏蔽底层资源的出错问题[62,63]。

图 9-2 是一个典型的云计算架构的示意图。从图中可以看出，用户端只需要简单的终

端设备(如个人计算机、笔记本电脑、PDA、手机等)就可以通过 Internet 访问到"云"中超大规模的计算和存储资源。用户管理层是用户与"云"交互的接入口,用户通过用户管理层接入到"云"中,云计算服务提供商通过用户管理层对用户进行接入并进行访问控制管理等。服务层将云中的各种资源管理起来并组织成相应的服务(比如存储、软件即服务、平台即服务等)提供给用户。虚拟资源层是将经过虚拟化的资源进行整合管理,然后提供给服务层以供分配。资源虚拟层是使用虚拟化技术对各种计算资源进行虚拟化处理,使得用户不必再关心真实物理机的位置、维护等问题。在云计算架构的底层是物理资源层,它是整个云计算平台的基础,存放着真正的物理资源。通过引入虚拟化,云计算使得用户不必再关心对物理主机的维护、管理以及优化问题了,从而把用户从繁琐、复杂的计算机资源管理中解脱了出来。

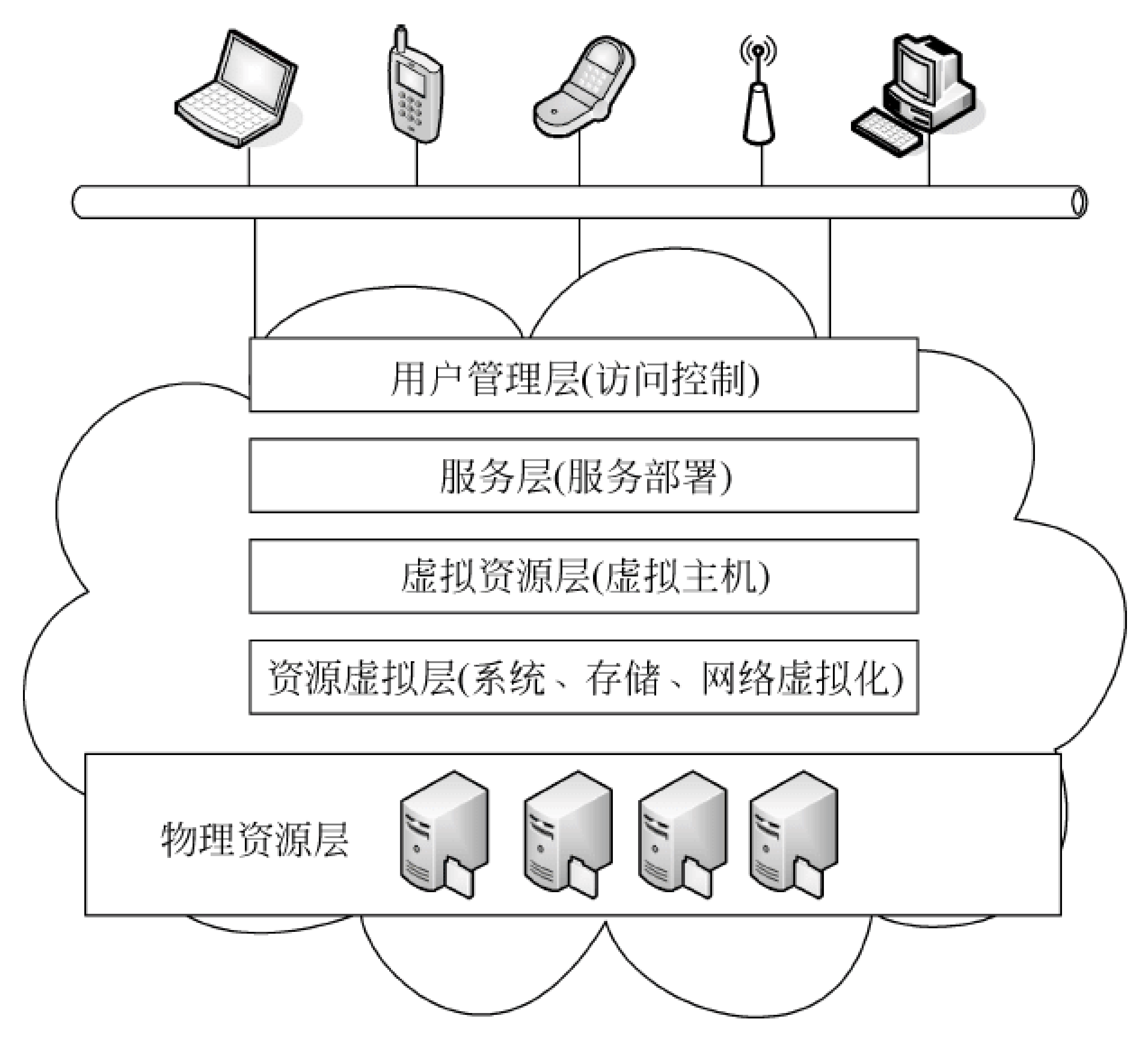


图 9-2　云计算架构

因为云计算巨大的商业前景,目前各大计算机巨头公司纷纷加入到云计算的行列中来,比如 Microsoft、Google、Amazon、IBM 等主流 IT 企业都开始提供云计算服务平台了。比如 Google 公司的 Google App、Microsoft 公司的 Azure、Amazon 公司的 EC2 和 S3、IBM 的 Blue Cloud 等。表 9-1 中总结了各大公司和他们推出的云计算产品及相关的研究项目。

表 9-1　云计算提供商及其产品

"云"名称	提供商	硬件环境	软件环境	提供服务类型
EC2	Amazon	x86	N/A	Web 服务
Blue Cloud	IBM	Blade Center 和 Xen 集群	powerVM 和 Hadoop	IT 服务部署
Google Apps	Google	Google Cluster	MapReduce BigTable	在线应用
Cloud Ware	3Tera	x86	Linux	虚拟数据中心

续表

“云”名称	提供商	硬件环境	软件环境	提供服务类型
EUCALYPTUS	UCSB	x86	Linux/Xen	虚拟主机与服务租赁
Network. com	SUN	x86	SunGrid Compute utility	高性能计算
SkyDrive	Microsoft	N/A	Windows Live	在线存储
XDrive	AOL	N/A	Windows Live	在线存储
VCloud	VMware	x86	VMware	虚拟数据中心
Daoli	EMC	x86	Xen and CHAOS	云计算研究

目前阶段的云计算主要提供以下服务[64]：

(1) 软件即服务(Software as a Service，SaaS)：目前使用的软件基本上都安装并运行于个人或单个组织的计算机中，软件的安装、版本升级、补丁、维护等都必须要由用户自己来进行，而且用户物理位置的改变需要将计算资源复制携带，所以移动性很差。云计算提供的软件即服务使得用户可以在网络环境下不受时间和地理位置限制，使用软件提供的服务，同时用户不必关心与服务无关的软件管理与维护问题。“Google App”就是一个典型的例子。

(2) 平台即服务(Platform as a Service，PaaS)通过 Internet 将操作系统和相关服务发送给客户，而客户并不需要下载或安装就可以使用，它是软件及服务的进一步扩展。平台即服务形式的云计算是将开发环境作为一种服务来提供给用户。使用户可以使用云计算服务提供商的开发环境来开发自己的程序，然后通过互联网和服务器再回传到用户处。从节省成本的角度考虑，中小企业可以通过租赁的方式从云计算平台中获得所需要的资源。客户只需要利用云计算提供的 PaaS 就能够建立、检测和部署自己的应用程序，不再需要购买硬件和软件，与传统的计算模式相比，PaaS 的最大的优势就在于开发相同的应用所需要的花费要节省很多。

(3) 基础设施服务(Infrastructures as a Service，IaaS)将基础设施出租，这主要归功于虚拟计算机的快速计算能力和稳定的存储能力。云计算的基础设施服务层相当于提供了一个中间的操作系统，它屏蔽了底层操作系统之间的异构。通过 IaaS，用户可以远程访问自己需要的计算资源，这些资源可以是计算、存储或其他资源，而不需要考虑对所获得的计算资源付出相应的基础 IT 软硬件的维护，升级等费用。

随着云计算的广泛应用，云计算平台的安全性成为云计算的核心问题之一，它主要有以下几方面[66~68]：

(1) 用户数据的安全性。著名的 Garnter 公司在一份关于云计算风险的调查报告中提出：“云计算平台服务提供商权利过大，致使云计算平台中的用户数据可能会被非法访问与泄露[69]”。另外，来自外部的恶意攻击也对数据的安全性构成了威胁，2005 年著名的在线付费提供商 PayMaxx 泄露了多于 250 000 个用户的信息，就是因为其系统的安全漏洞收到了来自外部的恶意攻击[70]。

(2) 云计算平台的容错性。云计算平台必须确保为其用户提供正确可用的服务。所以其平台内部软硬件故障的发生，不能对所提供的服务产生影响。

(3) 平台的可维护性。管理与维护超大规模的硬件和软件，这对于云计算平台来说，是很具有挑战性的，并且平台内部对软件和硬件的管理维护操作不能影响云计算平台上层所

提供的各种服务的正常运行。

在目前的云计算环境中，用户与云计算平台服务提供商之间的关系是不可靠的，这给云计算平台的安全带来了很大的危害。一方面，云计算用户想要确保他们的服务在云计算平台中被正确地运行，他们的数据信息相对于其他用户或组织是不可见的，而且不会被恶意使用；另一方面，云计算平台中存在了多个用户，它必须得保证少部分用户对平台的不安全操作不能影响到云计算平台本身，造成云计算平台对其他用户无法提供正常的服务。另外一个问题是目前日常使用的操作系统从本质上来说是不安全的，缺陷与安全漏洞的出现是不可避免的[71,72]。而且目前的操作系统是单内核设计，这使得某一部分的安全漏洞可能会引起整个操作系统被攻击。操作系统的接口对于应用程序的开放性，没有遵守最小特权原则(Principle of Least Privilege，POLP)，导致了操作系统被恶意用户控制的潜在威胁[73]。云计算提供了平台即服务，这增加了操作系统的复杂性，所以传统的安全策略，比如访问控制、完整性保护、平台度量等策略[74~76]已经不足以为用户的数据文件提供足够的安全性。

9.2.2 研究现状

云计算技术上的研究主要包括 4 个方面：云计算相关理论、云计算相关技术、云计算平台和云计算应用。

在云计算技术领域里，其中云计算编程方面，有很多研究机构开发了新的编程模式，对 MapReduce 编程模式进行扩展或者更新。Yahoo 扩展了 MapReduce 框架，在 MapReduce 步骤之后加入一个 Merge 的步骤，从而形成一个新的 MapReduceMerge 框架。使用这样的框架应用程序开发人员可以自己提供 Merge 函数，做两个数据集合的合并操作。斯坦福大学的研究人员将 MapReduce 的思想应用到多核处理器上，主要工作是在多核处理器的基础上构建了一套 MapReduce 的编程框架，并结合各种不同的应用程序在多核上的表现与现有的使用 pthread 编程方式进行比较。结果表明，在适合 MapReduce 表达的应用程序上，MapReduce 效率较高，在多核上的应用是有价值的。Wisconsin 大学的研究人员在 Cell 处理器上运行了基于 MapReduce 的应用程序。由于 Cell 处理器是异构多核的处理器，由一个中央处理器和八个协处理器构成，对此编程比较困难。他们将 MapReduce 的框架移植到 Cell 处理器的架构上。实验结果表明，Cell 处理器上的 MapReduce 程序有一定程度的性能提高。在不同于 MapReduce 编程方面，HP 的 Sinfonia 将注意力关注于分布式共享内存的使用。Sinfornia 提供了一个新的编程接口，一个对于内存的读写操作三元组“(Compare，Read，Write)”。在这个三元组中，Compare 是比较列表，由应用程序提供一系列的值与相应的集群内存中的数值进行比较，类似地，Read 和 Write 是读出和写入的列表，表明一系列的读写操作。其语意是首先进行 Compare 列表的比较，如果所有的比较都能得到满足，则进行三元组中的读写操作。如果上述的任何一个部分操作失败，则整个操作回卷到操作之前的状态，保持系统一致。目前已经在这种模式上完成了分布式文件系统的构建以及分布式的垃圾收集系统等。同时，这样的一种系统也能够容忍大量结点的失败，完成对于可用性的保证。

云计算具备最基本的三个环节：分布计算结点、计算能力集群和再分配的云、计算能力的用户结点。这三个环节中的任意一个出现安全问题，或者任意两个的连接处出现安全问题，都有可能带来整个云的崩溃。当前云计算安全研究机构主要集中于用传统安全的方法

解决云计算的安全问题，例如传统的加密、数字签名、访问控制等，而传统的安全方法面对大规模数据处理的云计算性能都得不到很好的保障。

9.2.3　产业状况

按照计算机行业的观点认为：云计算将成为全球 IT 业恢复性增长背景和物联网方向下的行业主要趋势；全球 IT 业将进入恢复性增长阶段，Gartner 报告显示近 10 年全球 IT 支出下降 5.2%，而今后 10 年全球 IT 支出预计增长 3.3%，国内 IT 业则从下游行业的旺盛需求及软件外包的高速增长获得支撑；IDC 预测，云计算相关 IT 支出 2012 年将达 423 亿美元，年复合增长率 27.3%，超出云计算以外 IT 支出部分增速(5.3%)5 倍以上。

关于云计算产业状况详见表 9-2。

表 9-2　国外云计算产业状况

云　计　算	描　　述	优　　点	缺　　点
亚马逊基础云计算 (Amazon Infrastrure Cloude)	基础服务内容： 弹性云计算(EC2) 简单存储服务(s3) 简单队列服务(sos) 简单队列服务(DB)	随时可用，Amazon.com 提供的共享 Web 级别的基础服务，真正的到期即付的使用模式，标准的 Web 服务界面、智能型语言	不支持 IDE 有限的服务水平 刚刚进入企业市场
Google 应用引擎 (Google App Engine)	紧密整合的开发与主机环境，用于 Web 上的应用：动态 runtime、持久存储、用户授权、电子邮件、服务监控、登录分析等	已有的、成熟的 Web 级别的基础服务；成套丰富的 Google 服务应用程序界面；入门门槛低(免费使用)	需要驱动、基于页面模式；不能直接访问 Google 的底层；没有服务水平协议
IBM 蓝色云计算 (IBM Blue Cloud)	用于构造私有云计算的硬件与软件集合：刀片机中心(带基于 Linux 的服务器、网络技术引擎、Xen 和 PowerVM)、Hadoop 和 Tivoli	在办公支持以及开发环境领域的强有力地位 开源组件 可自定义	供应特定的硬件，复杂，高成本
微软视窗 Azure (Microsoft Windows Azure)	软件+服务的混合模型，通过一套核心在线服务(比如存储、身份认证、数据服务)达到相同的功能，为桌面客户端及云计算提供相同的应用程序界面	投资规模大，视觉感官完整，大型开发者社区，现有可用的开发工具和技术，整合更容易，特别是与微软的其他应用程序的整合	进入市场时间有限，因可选商务模式多样而复杂
销售力量 (Force.com)	主机开发平台，从 AppExchange 发展出的新的工具，比如 visualforce(为客户定制的用户界面)以及 Apex Code(与 Java 类似的编程语言，用于处理商务逻辑和数据)	核心 CRM 应用程序的现有用户群，多用户结构，相对较成熟	独有的语言(Apex Code)，可扩展升级但非 Web 扩展的基础结构

续表

云 计 算	描 述	优 点	缺 点
虚拟机软件 v 云计算（Vmware vCloud）	虚拟的数据中心操作系统，包括一套有代表性的、基于状态传输的应用程序界面，用于管理云计算环境下的虚拟机，以及一整套服务（vServices），比如反向负载以及服务层次协议追踪	在云计算数据中心内，支持现场虚拟基础服务与外部 Vmware 之间的移动连接	目前只是与一系列服务伙伴合作的初步产物，何时实现，尚为可知

9.2.4 云计算的体系结构

本节将重点研究云计算的体系结构，并且分别对体系结构中的主服务控制机群、存储结点机群、应用结点机群、计算结点机群、输入设备和输出设备进行相关的研究。

云计算机是基于传统计算机的设计思想和云计算理念而生的一个新的体系结构。在云计算体系中应该具有 6 个组成成分：主服务控制机群、存储结点机群、应用结点机群、计算结点机群、输入设备和输出设备，如图 9-3 所示。

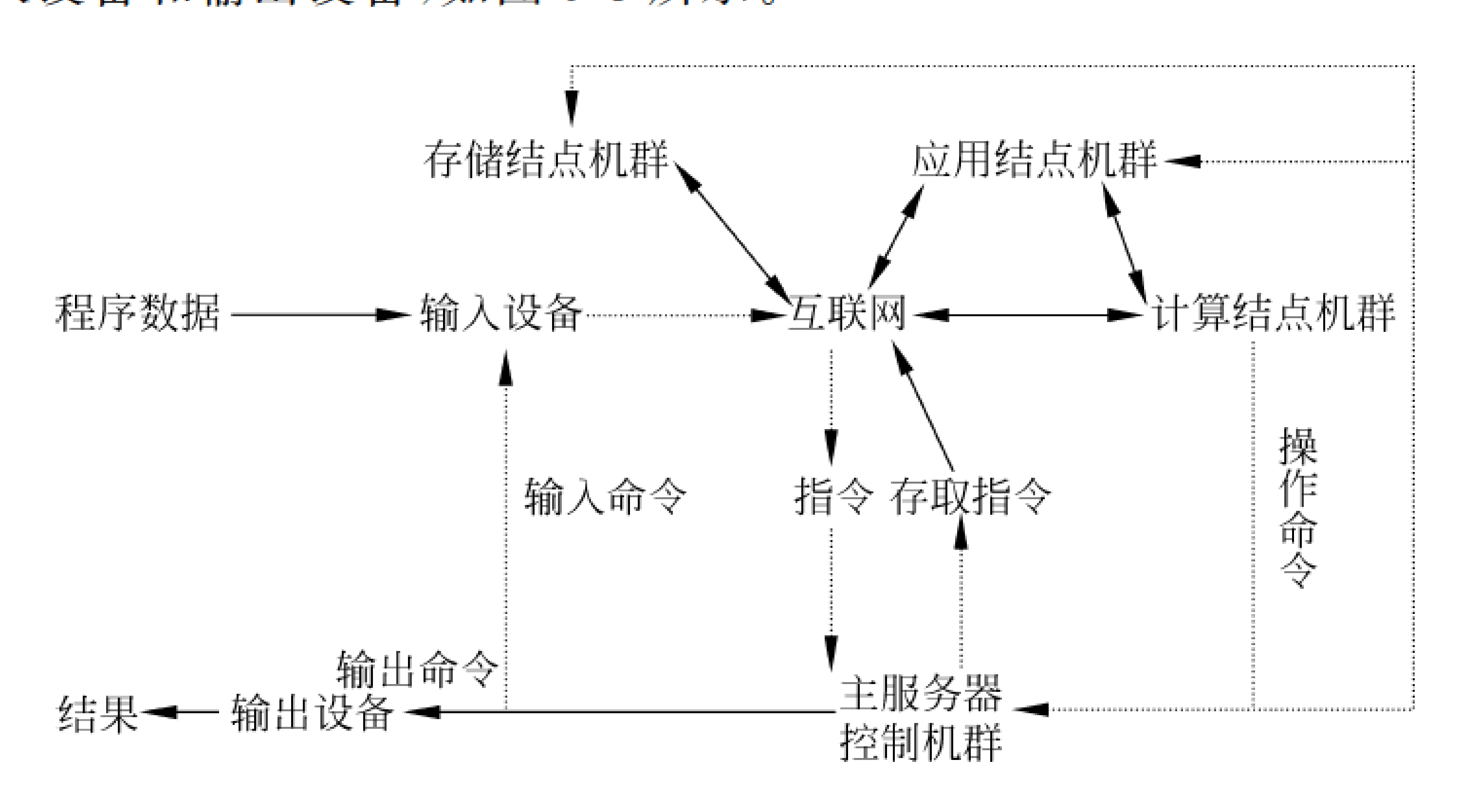


图 9-3 云计算通用体系结构

(1) 主服务控制机群对应于传统计算机体系结构中，可以类似看作控制器的部分。它是由一组或多组主引导服务机群和多组分类控制机群所组成的机群系统。主要负责接收用户应用请求、验证用户合法性，并根据应用请求类型进行应用分类和负载均衡。

(2) 存储结点机群和应用结点机群相当于传统计算机体系结构中的存储器部分，但又有所区别。存储结点机群是由庞大的磁盘阵列系统或多组拥有海量存储能力的机群系统所组成的存储系统，它的责任是处理用户数据资源的存取工作，并不关心用户对这些数据要如何应用，也不会处理存取数据资源和后台安全策略管理以外的任何操作。这里同时提出了一个新的概念——云盘，所谓云盘就是由云端主服务控制机群为云脑用户所分配的、建立在存储结点机群上的存储空间，它虽不是用户本地硬盘，但却完全由用户进行应用和管理，操作感与本地硬盘一致。应用结点机群则是由一组或多组拥有不同业务处理逻辑的机群系统

所组成的应用系统，它负责存储应用程序和处理各种逻辑复杂的用户应用。这两个结点机群是完全按照主服务控制机群的任务控制流程运行的，其本身不能拥有系统流程控制权。

(3) 计算结点机群提供类似运算器的功能。计算结点机群是由多组架构完善的云计算机群所组成，其主要工作是处理超大运算量要求的计算，并不提供小计算量服务。因为机群运算会在多级交互以及计算分配与组装上花费不少时间，所以小计算量运算如在计算结点机群进行处理不但开销大，而且很有可能效率远不如单机运算，可以说得不偿失。这些小计算量运算服务只需在应用结点机群或计算结点机群的某台机器中完成即可。

通过这样的体系结构，便可搭建出一个普通的云计算群，这仅是一套通用基础体系结构，针对不同应用将会有更细致的设计与部署、安全及备份，为第三方企业级开发提供 API 等也将会在未来实际开发应用中重点考虑。

我们将改良通用体系结构并设计一个复杂的云计算模型系统，考虑到基于机群系统开发的实际情况，我们将设计一个由 50～100 台刀片服务器组成的云计算机群模型，以便引导后续实习者或学习者进行实际开发。

在主服务控制机群的地方，为了简化系统的复杂度，将引入监控管理服务器的概念。在负载均衡时，只需取出监控管理服务器对各结点的状态监控信息，便可进行应用或存储策略分配。同时，将应用结点机群和存储结点机群合并为一个应用及存储结点机群，一是在实验中可节省成本和减少硬件资源的浪费；二是由于在实验开发环境中，整个系统架构在局域网中，网络带宽还不会成为系统的瓶颈，这样设计理论上可以提高网络负载，更容易暴露系统问题，易于发现问题、测试。

9.2.5　虚拟化逻辑协议

我们将重点研究云计算虚拟化逻辑协议以及云计算中的虚拟化技术原理。

虚拟化技术是一种逻辑简化技术，实现物理层向逻辑层的变化。从这个定义来看一个系统采用虚拟化技术后其对外表现出的运动方式是一种逻辑化的运动方式，而不是真实的物理运动方式。所以采用虚拟化技术能实现对物理层运动复杂性的屏蔽，使系统对外运行状态呈现出简单的逻辑运动形态，如图 9-4 所示。

虚拟化技术是一项综合性的技术，对于计算机科学来说大部分工作都是在做虚拟化的工作，网络的七层协议是对物理通信的虚拟化，传统的操作系统是对单个计算机物理硬件的虚拟化，计算机高级语言是对机器语言的虚拟化，人工智能则是更高级的虚拟化技术。计算机科学的虚拟化层次用图 9-4 进行了简单的描述，结点的物理硬件和网络物理硬件通过多层虚拟化的逻辑简化过程形成了弹性化的计算、存储和网络带宽三者整合的虚拟资源池，也就是云计算模式。

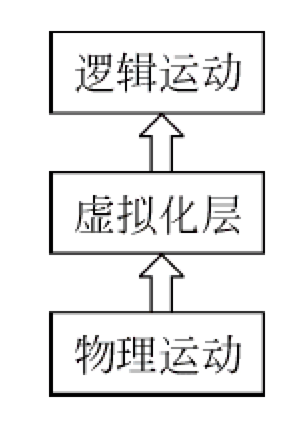


图 9-4　虚拟化技术的一般性逻辑

从图 9-5 可以看出云计算的概念位与整个逻辑的最上层，是底层物理硬件经过多次虚拟化抽象而形成的一个逻辑概念。这也就说明了为什么云计算是未来几年的技术发展方向，因为技术的发展就是一个不断抽象简化的过程，越上层的技术逻辑就是越高级的逻辑，因为这个逻辑是站在了很多巨人肩上的，这些巨人就是下层的虚拟化技术。

所以可以说云计算系统是运行在一个多层虚拟化协议栈上的高度抽象系统。

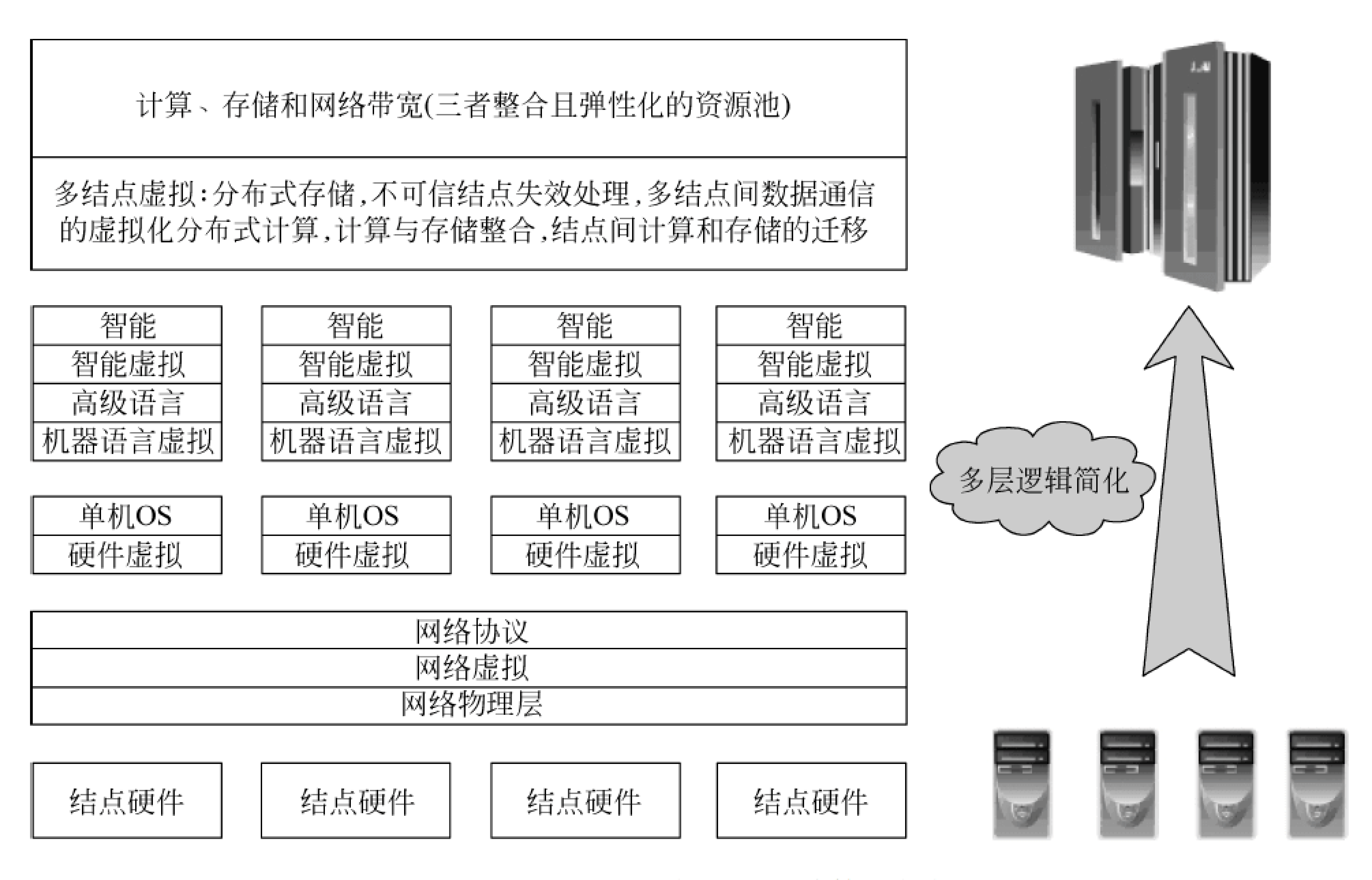


图 9-5　从虚拟化角度看云计算的架构

云计算标准就是要建立一个虚拟化的逻辑协议栈,使产业链上的不同企业能有效地在同一个逻辑递推关系下工作。图 9-6 是云计算协议栈的一个简图,其中的每一层可能还会包括许多子层,云计算概念是下层多个逻辑协议层递推出的一个上层概念。

云计算
多结点操作系统虚拟化逻辑协议层
多结点计算虚拟化逻辑协议层
多结点存储虚拟化逻辑协议层
单机操作系统虚拟化逻辑协议层
网络硬件虚拟化逻辑协议层
单结点硬件虚拟化逻辑协议层

图 9-6　云计算简化协议栈

9.2.6　海量数据高速处理算法

海量数据处理是云计算发展的必然结果,对海量数据分析和挖掘也显得越来越重要,从海量数据中提取有用信息重要而紧迫,这便要求处理要准确,精度要高,而且处理时间要短,得到有价值信息要快,所以,对海量数据的研究很有科学研究价值,值得我们进行广泛深入的研究。基于海量数据的数据挖掘正在逐步兴起,面对着超海量的数据,一般的挖掘软件或算法往往采用数据抽样的方式进行处理,这样的误差不会很高,大大提高了处理效率和处理的成功率。因此我们将重点研究海量数据处理,包括海量数据排序、海量数据查询、海量数据挖掘,海量数据去重统计等。

在实际的工作环境下，许多人会遇到海量数据这个复杂而艰巨的问题，它的主要难点有以下几个方面：

1. 数据量过大，数据中什么情况都可能存在

如果数据上到千万级别，甚至过亿，人为的每条去逐一检查已不可能，必须通过工具或者程序进行处理，尤其海量的数据中，什么情况都可能存在，例如，数据中某处格式出了问题，尤其在程序处理时，前面还能正常处理，突然到了某个地方问题出现了，程序终止了。

2. 软硬件要求高，系统资源占用过高

对海量的数据进行处理，除了好的方法，最重要的就是合理使用工具，合理分配系统资源。在云计算平台中，对云平台的并发计算要求较高。

9.2.7　虚拟化技术

云计算中最核心的技术就是虚拟化技术，它通过使用软硬件分时服务，模拟与仿真执行等技术，达到在单个计算机物理设备上模拟出多个相互隔离的硬件执行环境的目的。这个相互隔离的虚拟执行环境也被称为虚拟机（Virtual Machine，VM），在虚拟机上用户可以运行多个操作系统和各种应用程序。在位于虚拟机中运行的操作系统和底层硬件资源之间有一个称为虚拟机监控器（Virtual Machine Monitor，VMM）的附加层。它专门负责管理底层硬件资源，然后将其分配给上层运行的虚拟机。

根据其提供的平台类型，将 VMM 分为两类：完全虚拟化和部分虚拟化。

1. 完全虚拟化

完全虚拟化（Full Virtualization）方式中 VMM 虚拟的是真实的物理平台，在操作系统看来，完全虚拟化方式的虚拟平台与真实的物理平台区别不大，根本感觉不到是运行在虚拟的平台之上。这样的虚拟平台不需要对操作系统作修改就可以直接运行，所以称为完全虚拟化。

完全虚拟机中所虚拟的是相应的物理硬件的完全逻辑复制。它独立于其他虚拟机，而且完全虚拟机之间也是互不相见的，就像是现实中的一个独立完整的的计算机设备。所以在完全虚拟机中运行的操作系统与在真实的物理设备上的运行几乎是一样的。而且完全虚拟机不需要对操作系统做任何修改。操作系统对虚拟出来的设备，比如：虚拟处理器，虚拟内存及虚拟 I/O 设备的操作与相应的物理设备的操作是一样的。运行在 IBM360/67 上的 CP-67 系统是最早实现完全虚拟化的。

最流行的虚拟化方法使用名为 hypervisor 的一种软件，在虚拟服务器和虚拟化技术底层硬件之间建立一个抽象层。VMware 和微软的 Virtual PC 是代表该方法的两个商用产品，而基于核心的虚拟机（KVM）是面向 Linux 系统的开源产品。hypervisor 可以捕获 CPU 指令，为指令访问硬件控制器和外设充当中介。因而，完全虚拟化技术几乎能让任何一款操作系统不用改动就能安装到虚拟服务器上，而它们不知道自己运行在虚拟化环境下。主要缺点是，hypervisor 给处理器带来开销。在完全虚拟化的环境下，hypervisor 运行在裸硬件上，充当主机操作系统；而由 hypervisor 管理的虚拟服务器运行客户端操作系统（guest OS）。

2. 准虚拟化

完全虚拟化是处理器密集型技术，因为它要求 hypervisor 管理各个虚拟服务器，并让它们彼此独立。减轻这种负担的一种方法就是，改动客户操作系统，让它以为自己运行无法虚拟 64 位客户操作系统。在虚拟环境下，能够与 hypervisor 协同工作。这种方法就叫准虚拟化(Para-Virtualization)，准虚拟化也称部分虚拟化，它是通过修改操作系统内核的源代码以规避部分无法实现虚拟化指令的方式，使得虚拟机监控器(VMM)可以将物理资源进行虚拟化。完全虚拟化中无法处理的特殊指令是通过 Binary Translation 将相应的客户机二进制代码进行修改处理的。而类虚拟化采用了完全不同的思路，它是通过修改 API 级的操作系统内核代码，使得操作系统内核完全规避了那些无法虚拟化的指令。

Xen 是开源准虚拟化技术的一个例子。操作系统作为虚拟服务器在 Xen hypervisor 上运行之前，它必须在核心层面进行某些改变。因此，Xen 适用于 BSD、Linux、Solaris 及其他开源操作系统，但不适合对像 Windows 这些专有的操作系统进行虚拟化处理，因为它们无法改动。准虚拟化技术的优点是性能高。经过准虚拟化处理的服务器可与 hypervisor 协同工作，其响应能力几乎不亚于未经过虚拟化处理的服务器。准虚拟化与完全虚拟化相比优点明显，以至于微软和 VMware 都在开发这项技术，以完善各自的产品。

下面将重点介绍以下 4 项虚拟化核心技术：

1）CPU 和 MEM 的热插拔技术

云计算的基础设施服务是提供：计算能力、网络带宽能力、存储能力。计算能力的动态扩展，也就是虚拟 CPU 和 MEM 的热插拔技术是其中最难的部分，这不仅涉及硬件对热插拔的支持，还涉及虚拟化技术对热插拔的支持。使得其相应可以在我们的服务中真正实现虚拟机的动态扩容，实现真正的弹性，能在不停客户服务的情况下变更计算能力。因此我们将重点研究虚拟 CPU 和 MEM 的热插拔技术。

2）软交换技术

软交换技术，是用来提供虚拟机之间的虚拟网络数据通信。目前部分公司和科研机构也在研发软交换程序，但目前大都是二层交换。这些软交换设备通过上联到一个真实交换机上实现一个级联的交换网络，可以提供 VLan 划分，限速，流量控制等功能，但是效率不够高，我们将研究高效的软交换技术。

3）IAAS API

目前，虚拟化技术的两个主要发展方向是服务器虚拟化和桌面虚拟化，而桌面虚拟化是将整个 IT 带入云计算时代的关键，这意味着可以通过各种便携式设备访问云端的服务。而这些服务必须具备统一的访问协议、API。IAAS 需要为上层服务、第三方开发者提供统一的访问协议。因此，我们将重点研究 IAAS 的访问协议和 API 接口。

4）与分布式计算的结合

虚拟化技术是将计算能力进行“分”，使得每一个虚拟计算单元的能力小于支撑它的物理硬件。而目前稍微具备规模的应用其对计算能力的需求往往大于一个单独的物理硬件，这就需要分布式计算的支持，也就是“合”。“分”能带来很多好处，例如资源复用、方便的管理等，但如果只是“分”而不对“分”以后的资源有效的“合”，那么就只能满足小客户的应用，不能满足大客户的应用。因此，我们将重点研究如何有效地将虚拟化技术与分布式计算相结合。

9.2.8　云计算相关技术

1. 云服务器的研发

大型服务器是云计算的核心硬件设备，该产品的成功自主研发将为我国自主掌握云计算基础，实现国家信息安全可控，将发挥重要作用，具有长远的产业战略意义，对于解决当前由大到强的 IT 产业问题也具有基础性作用。

我们将研究采用紧耦合共享内存、硬件分区等高端服务器技术，按当前业界高标准研究设计支持 8 颗 64 核心处理器，1TB 的 DDR3 内存，能降低能源消耗，减少碳排放的云计算服务器。同时，将研究可靠性技术，如计算链路全冗余、故障自动切换技术以及多种可信技术，这样的服务器能满足金融、电信、能源、政府等国民经济关键行业对云计算信息化 7×24 小时不间断运行需求。同时，我们需要研究 BIOS 开发、高速互联背板设计等多项业界技术难题。

2. 云存储技术

为保证高可用、高可靠和经济性，云计算采用分布式存储的方式来存储数据，采用冗余存储的方式来保证存储数据的可靠性，即为同一份数据存储多个副本。另外，云计算系统需要同时满足大量用户的需求，并行地为大量用户提供服务。因此，云计算的数据存储技术必须具有高吞吐率和高传输率的特点。因此，在云存储技术方向上，我们将重点研究具有高吞吐率和高传输率的云存储技术。

云计算的数据存储技术未来的发展将集中在超大规模的数据存储、数据加密和安全性保证以及继续提高 I/O 速率等方面。以 GFS(Google File System)为例。GFS 是一个管理大型分布式数据密集型计算的可扩展的分布式文件系统。它使用廉价的商用硬件搭建系统并向大量用户提供容错的高性能的服务。GFS 和普通的分布式文件系统有以下区别，如表 9-3 所示。

表 9-3　GFS 与传统分布式文件系统的区别

文件系统	组件失败管理	文件大小	数据写方式	数据流和控制流
GFS	不作为异常处理	少量大文件	在文件末尾附加数据	数据流和控制流分开
传统分布式文件系统	作为异常处理	大量小文件	修改现存数据	数据库和控制流结合

GFS 系统由一个主控端和大量块服务器构成。主控端存放文件系统的所有元数据，包括名字空间、存取控制、文件分块信息、文件块的位置信息等。GFS 中的文件切分为 64MB 的块进行存储。

在 GFS 文件系统中，采用冗余存储的方式来保证数据的可靠性。每份数据在系统中保存 3 个以上的备份。为了保证数据的一致性，对于数据的所有修改需要在所有的备份上进行，并用版本号的方式来确保所有备份处于一致的状态。客户端不通过主控端读取数据，避免了大量读操作使主控端成为系统瓶颈。客户端从主控端获取目标数据块的位置信息后，直接和块服务器交互进行读操作。

GFS 的写操作将写操作控制信号和数据流分开，如图 9-7 所示。

即客户端在获取主控端的写授权后，将数据传输给所有的数据副本，在所有的数据副本都收到修改的数据后，客户端才发出写请求控制信号。在所有的数据副本更新完数据后，由

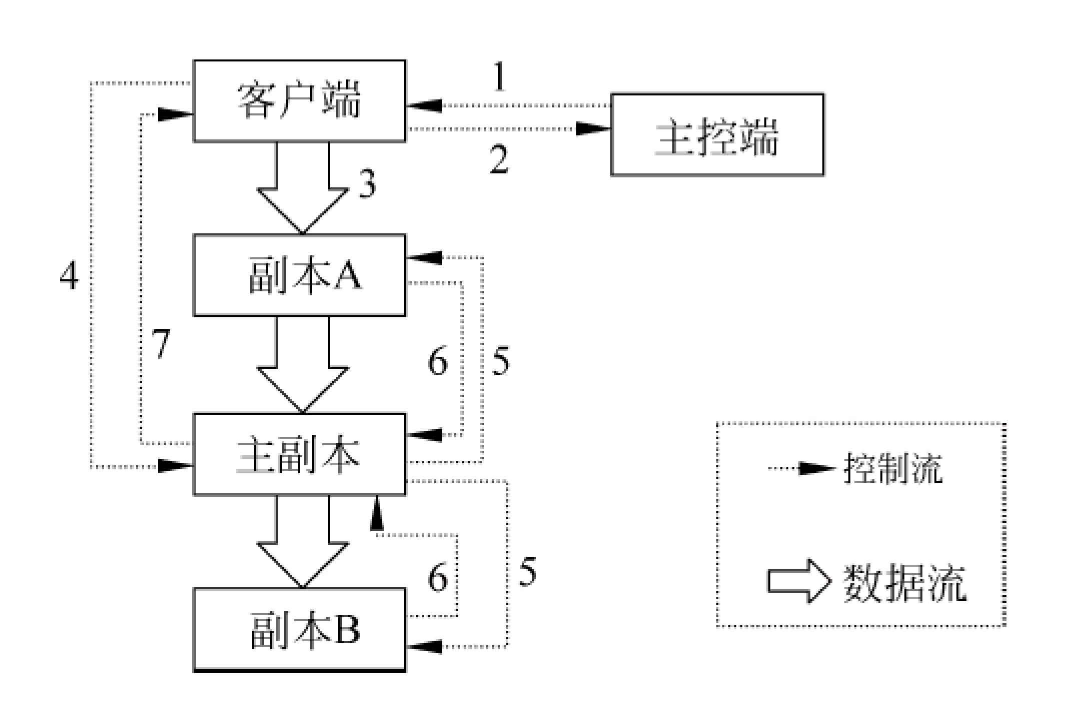


图 9-7 写控制流和写数据流

主副本向客户端发出写操作完成控制信号。

当然,云计算的数据存储技术并不仅仅只是 GFS,其他 IT 厂商,包括微软、Hadoop 开发团队也在开发相应的数据管理工具。本质上是一种分布式的数据存储技术,以及与之相关的虚拟化技术,对上层屏蔽具体的物理存储器的位置、信息等。快速的数据定位、数据安全性、数据可靠性以及底层设备内存储数据量的均衡等方面都需要继续研究完善。

3. 云数据管理技术

云计算系统对大数据集进行处理、分析向用户提供高效的服务。因此,我们将研究能高效地管理大数据集数据管理技术,包括大数据集的添加、删除、修改、备份等管理技术。其次,我们将重点研究如何在规模巨大的数据中找到特定的数据。我们将研究 Google 提出的 BigTable 数据处理技术,研究其优缺点以及改良其设计。

云计算的特点是对海量的数据存储、读取后进行大量的分析,数据的读操作频率远大于数据的更新频率,云中的数据管理是一种读优化的数据管理。因此,云系统的数据管理往往采用数据库领域中列存储的数据管理模式。将表按列划分后存储。

云计算的数据管理技术中最著名的是谷歌提出的 BigTable 数据管理技术。由于采用列存储的方式管理数据,如何提高数据的更新速率以及进一步提高随机读速率是未来的数据管理技术必须解决的问题。

以 BigTable 为例。BigTable 数据管理方式设计者 Google 给出了如下定义:"BigTable 是一种为了管理结构化数据而设计的分布式存储系统,这些数据可以扩展到非常大的规模,例如在数千台商用服务器上的达到 PB(Petabytes)规模的数据。"

BigTable 对数据读操作进行优化,采用列存储的方式,提高数据读取效率。BigTable 管理的数据的存储结构为:

<row:string,column:string,time:int64>->string。BigTable 的基本元素是:行,列,记录板和时间戳。其中,记录板是一段行的集合体,如图 9-8 所示。

BigTable 中的数据项按照行关键字的字典序排列,每行动态地划分到记录板中。每个结点管理大约 100 个记录板。时间戳是一个 64 位的整数,表示数据的不同版本。列族是若干列的集合,BigTable 中的存取权限控制在列族的粒度进行。BigTable 在执行时需要三个主要的组件:链接到每个客户端的库,一个主服务器,多个记录板服务器。主服务器用于分配记录板到记录板服务器以及负载平衡,垃圾回收等。记录板服务器用于直接管理一组记录板,处理读写请求等。

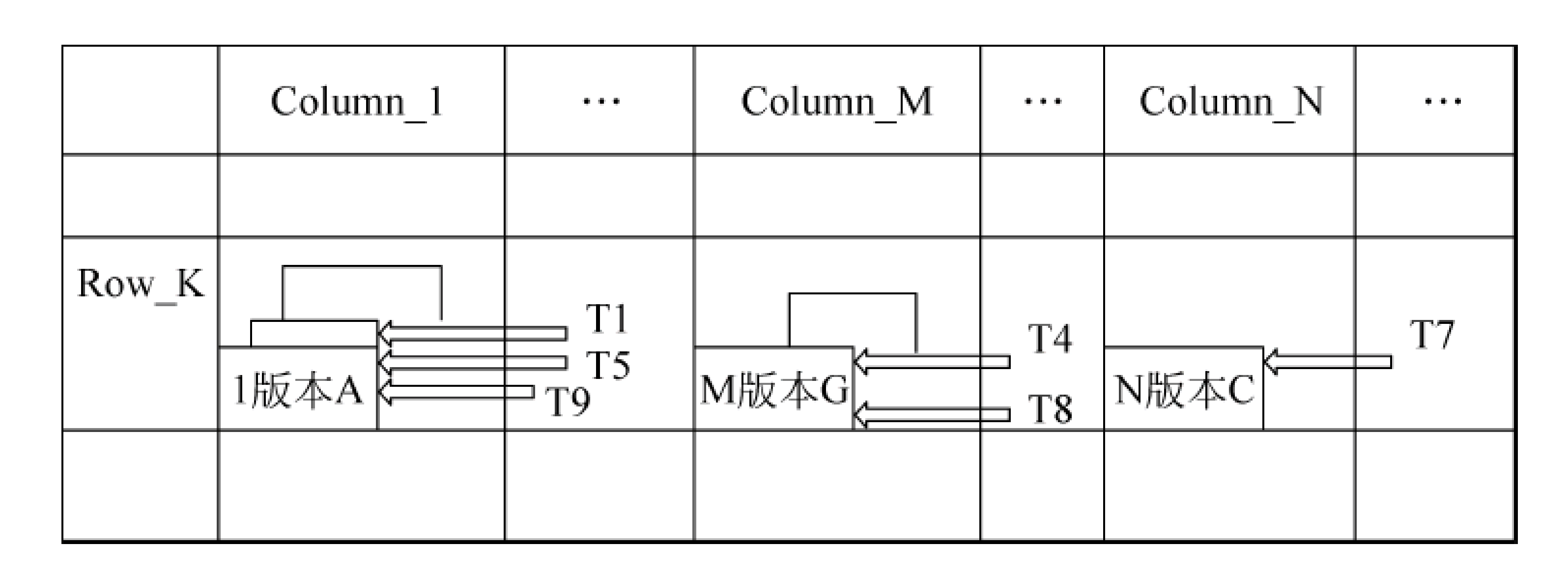


图 9-8　BigTable 的逻辑结构

为保证数据结构的高可扩展性，BigTable 采用三级的层次化的方式来存储位置信息，如图 9-9 所示。其中第一级的 Chubby File 中包含 Root Tablet 的位置，Root Tablet 根表单有且仅有一个，包含所有 Metadata Tablets 元数据表单的位置信息，每个 Metadata Tablets 包含许多 User Table 的位置信息。

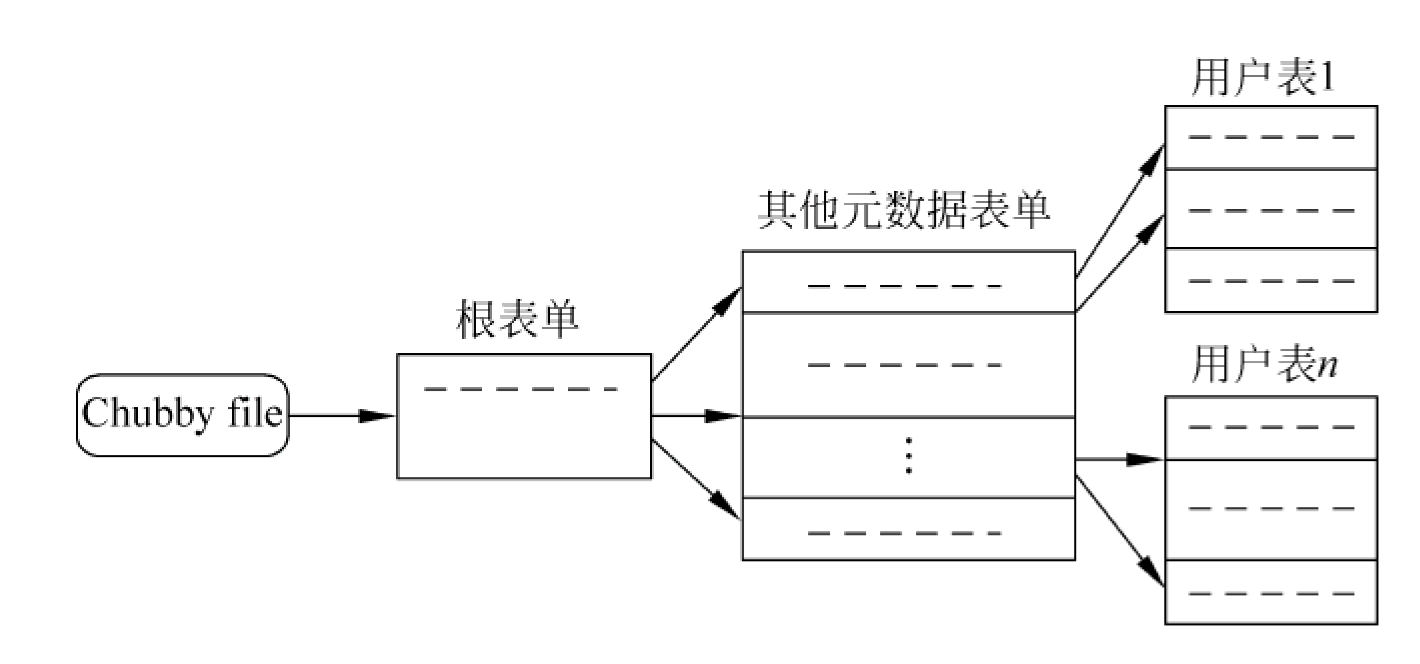


图 9-9　BigTable 中存储记录板位置信息的结构

当客户端读取数据时，首先从 Chubby File 中获取 Root Tablet 的位置，并从中读取相应 Metadata Tablet 的位置信息。接着从该 Metadata Tablet 中读取包含目标数据位置信息的 User Table 的位置，然后从该 User Table 中读取目标数据的位置信息项。据此信息到服务器中特定位置读取数据。

这种数据管理技术虽然已经投入使用，但是仍然具有部分缺点。例如，对类似数据库中的 Join 操作效率太低，表内数据如何切分存储，数据类型限定为 string 类型过于简单等。而微软的 DryadL INQ 系统则将操作的对象封装为. NET 类，这样有利于对数据进行各种操作，同时对 Join 进行了优化，得到了比 BigTable+MapReduce 更快的 Join 速率和更易用的数据操作方式。

4. 云计算的编程技术

我们将重点研究 Map/Reduce 编程技术，在其基础上提出我们自主的云计算编程模型。为了使用户能更轻松地享受云计算带来的服务，让用户能利用该编程模型编写简单的程序来实现特定的目的，云计算上的编程模型必须十分简单。必须保证后台复杂的并行执行和任务调度向用户和编程人员透明。

而当前云计算大部分采用 Map/Reduce 的编程模式。现在大部分 IT 厂商提出的“云”计划中采用的编程模型，都是基于 Map/Reduce 的思想开发的编程工具。Map/Reduce 不仅仅是一种编程模型，同时也是一种高效的任务调度模型。Map/Reduce 这种编程模型十

分适用于云计算。该编程模式仅适用于编写任务内部松耦合、能够高度并行化的程序。如何改进该编程模式，使程序员得能够轻松地编写紧耦合的程序，运行时能高效地调度和执行任务，是 Map/Reduce 编程模型未来的发展方向，也是我们实验室的重要研究方向。

然而基于 Map/Reduce 的开发工具 Hadoop 并不完善。特别是其调度算法过于简单，判断需要进行推测执行的任务的算法造成过多任务需要推测执行，降低了整个系统的性能。我们将研发改进型的 Map/Reduce 的开发工具，包括任务调度器、底层数据存储系统、输入数据切分、监控"云"系统等方面是将来我们研究发展的方向。

Map/Reduce 是一种处理和产生大规模数据集的编程模型，程序员在 Map 函数中指定对各分块数据的处理过程，在 Reduce 函数中指定如何对分块数据处理的中间结果进行归约。用户只需要指定 Map 和 Reduce 函数来编写分布式的并行程序。当在集群上运行 Map/Reduce 程序时，程序员不需要关心如何将输入的数据分块、分配和调度，同时系统还将处理集群内结点失败以及结点间通信的管理等。图 9-10 给出了一个通用 Map/Reduce 程序的具体执行过程。

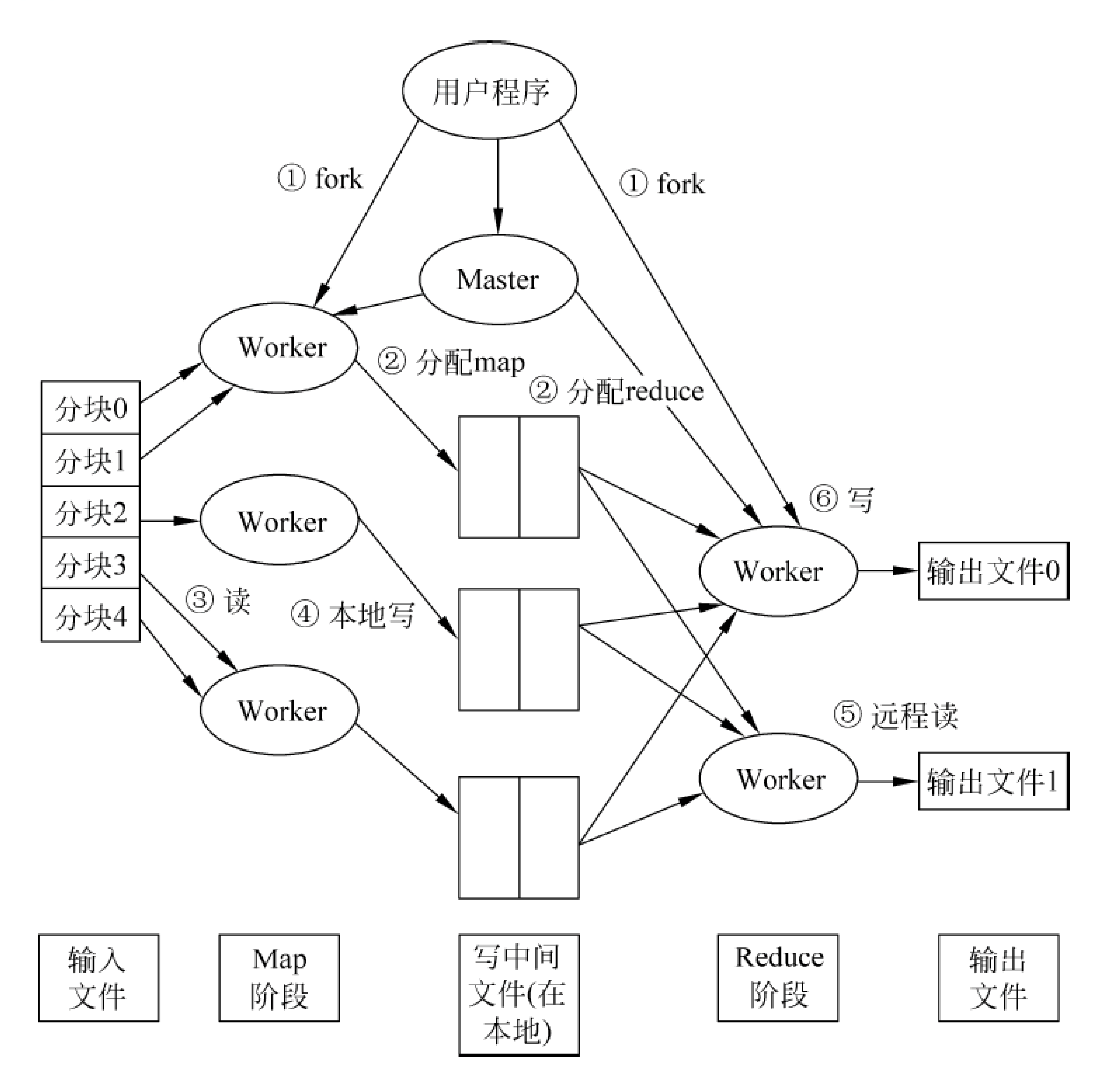


图 9-10　Map/Reduce 程序的具体执行过程

从图 9-10 可以看出，执行一个 Map/Reduce 程序需要 5 个步骤：输入文件、将文件分配给多个 Worker 并行地执行、写中间文件(本地写)、多个 Reduce Workers 同时运行、输出最终结果。本地写中间文件在减少了对网络带宽的压力同时减少了写中间文件的时间耗费。执行 Reduce 时，根据从 Master 获得的中间文件位置信息，Reduce 使用远程过程调用，从中间文件所在结点读取所需的数据。

Map/Reduce 模型具有很强的容错性，当 Worker 结点出现错误时，只需要将该 Worker

结点屏蔽在系统外等待修复，并将该 Worker 上执行的程序迁移到其他 Worker 上重新执行，同时将该迁移信息通过 Master 发送给需要该结点处理结果的结点。Map/Reduce 使用检查点的方式来处理 Master 出错失败的问题，当 Master 出现错误时，可以根据最近的一个检查点重新选择一个结点作为 Master 并由此检查点位置继续运行。

Map/Reduce 仅为编程模式的一种，微软提出的 DryadL INQ[17] 是另外一种并行编程模式。但它局限于.NET 的 LINQ 系统同时并不开源，限制了它的发展前景。

Hadoop(现已纳入 Apache 许可证之下)是一个在集群上运行大型数据处理应用程序的开放式源代码框架。它支持通过 Google 的 Map/Reduce 编程范例来创建并执行应用程序，该范例将应用程序划分为可以在集群中任何结点上执行的工作片段。它还通过一个分布式文件系统透明地支持可靠性和数据迁移。通过 Hadoop，“云”可以在合理时间内对庞大的数据集执行并行应用程序，从而支持计算密集型服务，比如高效检索信息，根据历史记录定制用户会话，或基于“蒙特卡罗”(概率)算法生成结果。

5. 云安全技术

迄今为止，安全性问题仍然是云存储潜在用户的最大顾虑。其中一个问题是用户缺乏对网络的控制，而网络是提供云存储和云计算资源的基础。于是，用户会怀疑数据被盗窃甚至被篡改。因此，对于任何登录 ID 和密码之类的敏感信息都应该实施保护。目前大部分邮件和 CRM 应用软件供应商都在云服务中提供这种基本安全性保护。

安全性一般包括四大元素，即身份验证、授权、访问控制和审计。

过去安全性设计是针对周边安全性，用以拒绝外部非授权用户的访问。在虚拟化环境中，虚拟 IT 服务不存在物理边界。于是，业务必须假设所有传输的数据都有潜在的被拦截风险。

系统取消了物理控制，就必须依靠其他加固原理来限制对信息的访问。信息的加密是其中最重要的方法，它可以限制对有用信息的访问，这种限制甚至超过了对物理系统的保护级别。于是，加密成为云服务安全性的关键性部分。

云存储所面临的问题尤其具有挑战，因为云存储中数据必须以某种加密格式保存和获取。一旦加密密钥本身丢失或损坏，那么数据也将丢失或损坏。因此，云存储和存储安全性比普通的云安全性问题更加具有挑战性。云安全应该具备以下能力：

(1) 用户结点首先要有威胁隔离能力，这里的隔离是指，当使用同一个云的邻居用户被入侵后，通过该邻居对本用户结点产生的威胁被隔离。假如 gmail 服务和 twitter 服务都使用同一个云，当 twitter 的 Web 服务虚拟主机被入侵后，想要通过 twitter 的 Web 服务虚拟主机再入侵 gmail 的数据库虚拟主机的企图不会得逞。

(2) 防御宿主的攻击，宿主机在种种安全措施下虽然很安全，但是万一也被入侵的话，通过虚拟 API 就可以对用户结点造成很大的伤害。因此用户结点在给宿主机很大控制权的情况下，要保留拒绝管理的能力。当然这样的防御策略比较难制定，但却不可或缺。

(3) 灾害隔离能力，这是指在用户结点被破坏后，要有自动隔离不让本区域的灾害蔓延到其他用户结点，这就有点类似家庭电力系统的保险丝，通过保险丝来防止自家电力问题蔓延到邻居家或整栋建筑物内。

6. 云搜索技术

云计算最早由 Google 公司应用到其搜索引擎上，使得搜索性能得到大幅提高，随着云

计算技术的发展，云搜索技术也逐渐得到重视。我们将重点研究云爬虫技术原理、海量数据抓取、海量数据解析、中文分词等关键技术问题。

如图 9-11 所示，传统的搜索引擎技术分为下载、分析、索引和查询 4 个系统，这 4 个系统相互配合，共同实现了搜索引擎的需求。下载系统负责从万维网上下载各种类型的网页，并保持与万维网的同步。

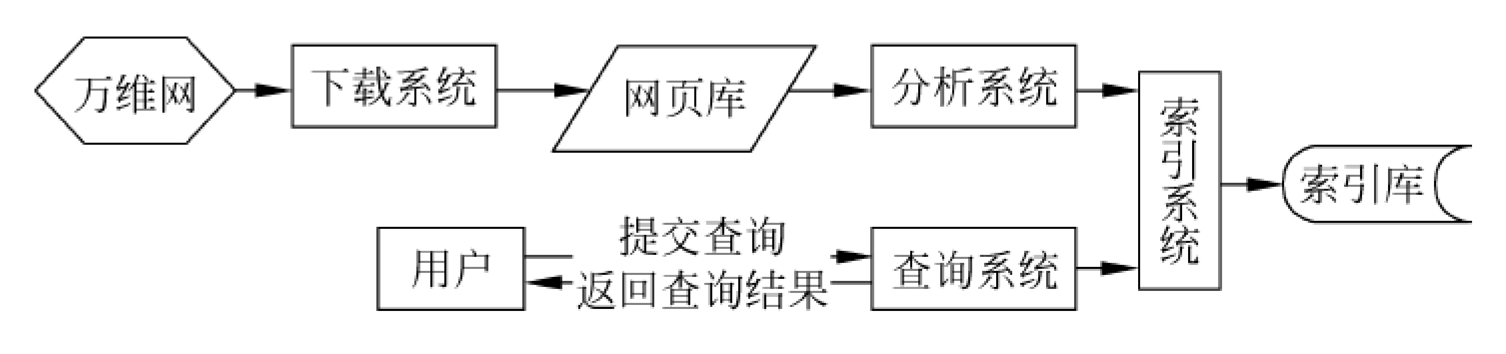


图 9-11 搜索引擎简单结构图

分析系统负责分析下载系统得到的网页数据，进行信息抽取、网页去重、中文分词和 PageRank 等。

索引系统负责将分析系统处理后的网页对象索引入库，作为搜索引擎的数据大本营，需要存储数以亿计的网页，并需要支持多用户的同时检索，提供低于秒级的检索时间。

查询系统负责分析用户提交的查询请求，经过检索、排序、提取摘要等相关操作，从索引库中检索出网页并将网页排序后，以查询结果的形式返回给用户。

图 9-12 是云搜索引擎系统结构，分为云爬虫、网页分析器、协调器、索引等。

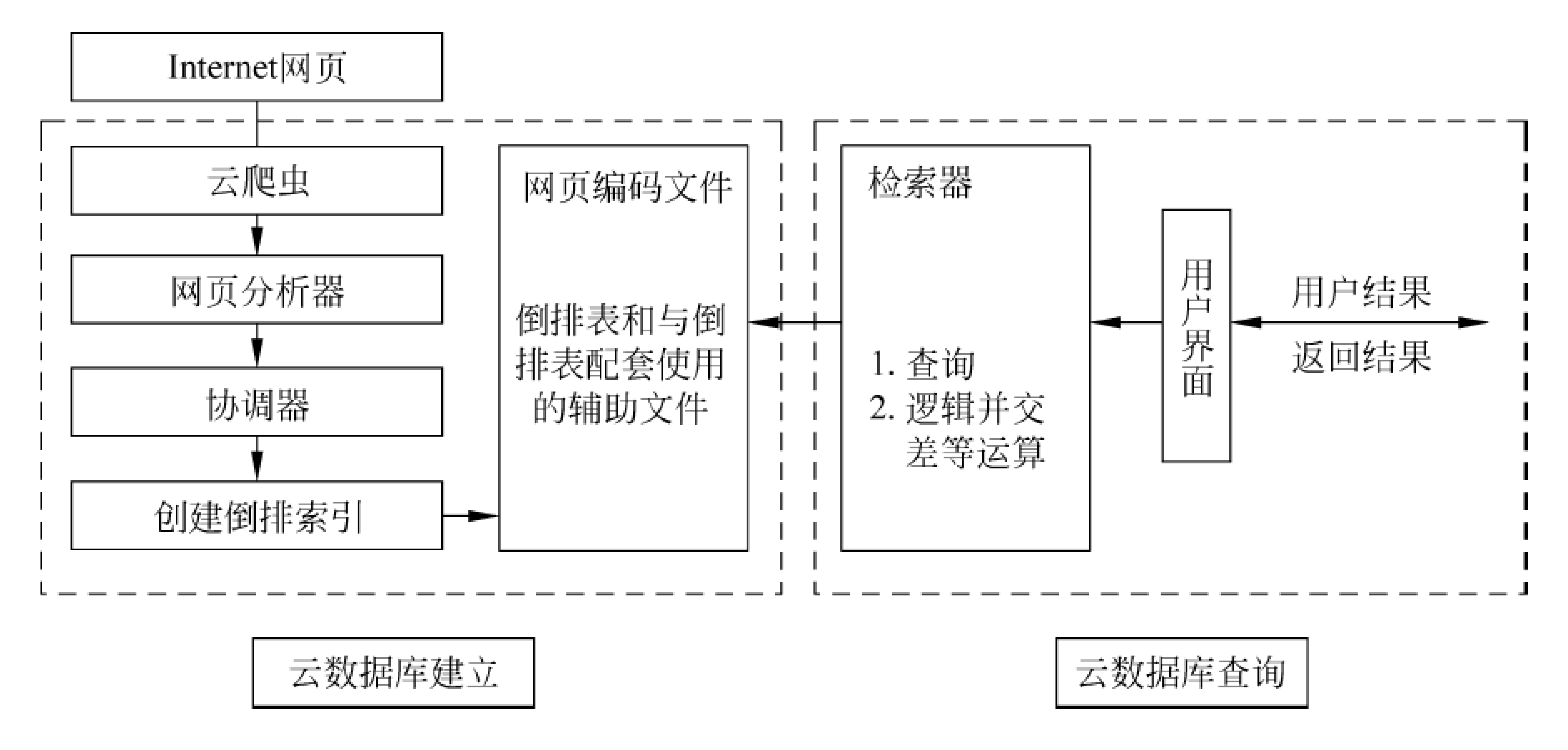


图 9-12 搜索引擎系统的结构

7. 云爬虫技术

云爬虫是一个功能很强的自动提取网页的分布式程序，其运行于云服务器中，它为搜索引擎从万维网上下载网页，是云搜索的重要组成。它通过请求站点上的 HTML 文档访问某一站点，遍历 Web 空间，不断从一个站点移动到另一个站点，自动建立索引，并加入到网页云数据库中。云爬虫进入某个超级文本时，它利用 HTML 语言的标记结构来搜索信息及获取指向其他超级文本的 URL 地址，可以完全不依赖用户干预实现网络上的自动爬行和搜索。

根据抓取过程云爬虫程序主要分为三个功能模块：一个是网页读取模块主要是用来读取远程 Web 服务器上的网页内容；另一个是超链分析模块，这个模块主要是分析网页中的

超链接,将网页上的所有超链接提取出来,放入到待抓取 URL 列表中;再一个模块就是内容分析器模块,这个模块主要是对网页内容进行分析,将网页中所有 HTML 标签去掉只留下网页文字内容。

云爬虫程序的主要工作流程如图 9-13 所示,首先云爬虫程序读取抓取站点的 URL 列表,取出一个站点 URL,将其放入未访问的 URL 列表(UVURL 列表)中,如果 UVURL 不为空则从中取出一个 URL 判断是否已经访问过,若没有访问过则读取此网页,并进行超链分析及内容分析,然后将此网页存入文档数据库,将 URL 放入已访问 URL 列表(VURL 列表),直到 UVRL 为空为止,此时再抓取其他站点,依次循环直到所有的站点 URL 列表都被抓取完为止。

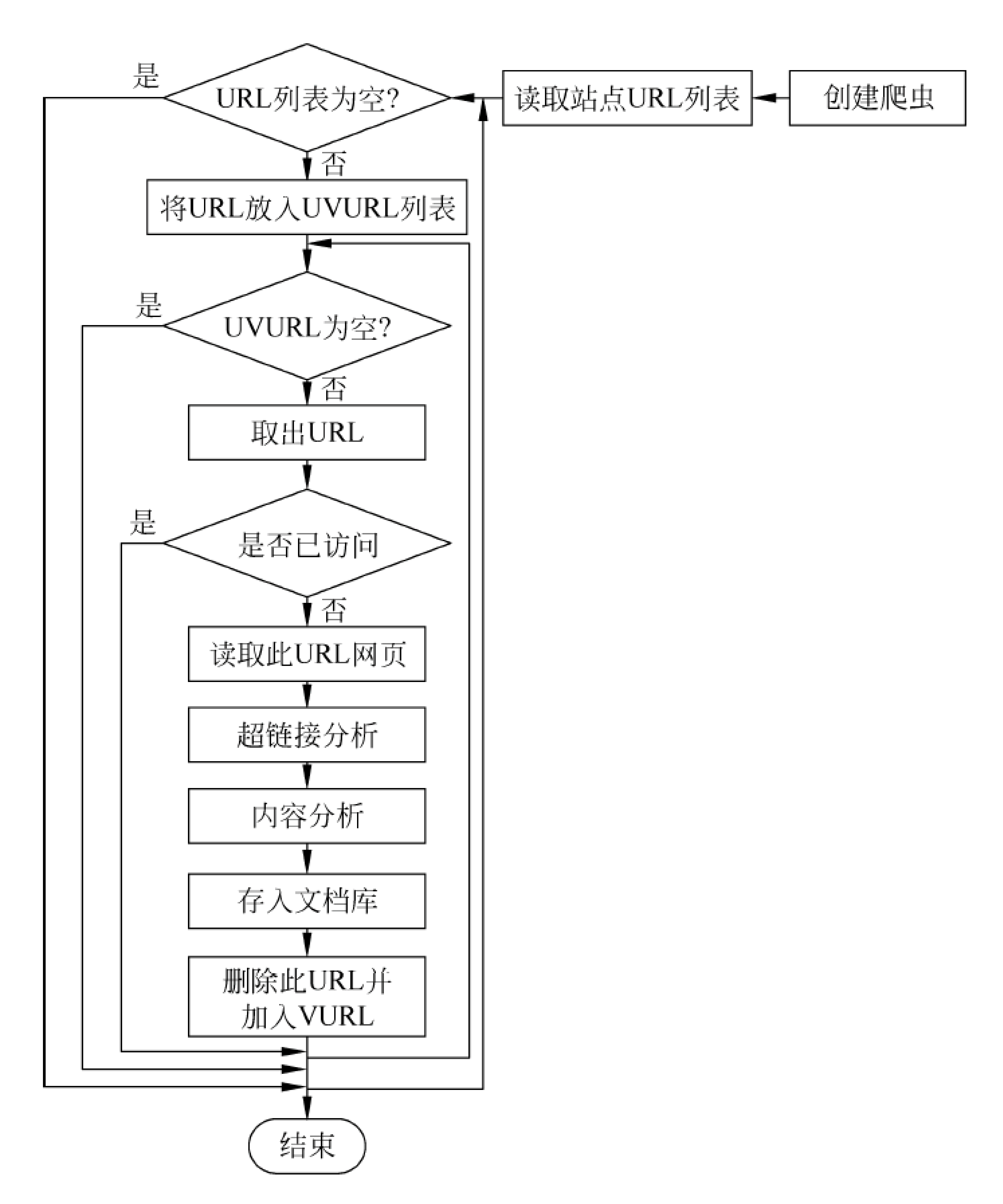


图 9-13　云爬虫程序工作流程

注:UVURL 是当前站点未访问的 URL,VURL 是当前站点已访问的 URL。

9.3　云计算平台的安全研究

9.3.1　云计算安全概述

云计算给信息技术领域带来了重大变革。这种变革为信息安全领域带来了巨大的冲击:

(1) 各类云计算应用没有统一不变的基础设施,没有统一的安全边界,很难实现用户数据安全与隐私保护;

(2) 云平台中数据与计算高度集中,安全措施必须满足海量信息处理需求;

(3) 云服务所涉及的资源由多个管理者所有,存在各种利益冲突,无法统一规划部署安全防护措施。

广义云计算的安全性包括云服务的保密性、完整性和可用性。

1. 保密性

保密性是指对数据或者信息的隐藏。安全的系统必须确保对数据的保密。数据保密的需求源自计算机在敏感领域的使用,比如政府或者企业。政府中的军事、民事机构经常对需要获得信息的访问者设定限制。云计算数据存储在数据安全中心,这就对数据的保密性提出更高的要求。各云计算使用者对数据进行保密处理主要关注以下方面内容:

数据通信隐私:必须确保通讯过程中数据的隐私。对于个人而言,隐私是指每个人应具有对其机密信息如医疗信息、工作经历和信用记录等机密信息的控制能力。在商业活动中,隐私包括了商业机密、产品设计信息、交易过程记录、竞争对手分析以及市场调查和销售计划等。对于军事部门而言,隐私包括了诸如在收集和分析人工统计信息的过程中需要保护的人员信息的问题,还包括了保护那些关系到军事武器的机密信息的问题。

数据的安全存储:必须确保敏感数据在存储过程中的完整性和隐私性,即敏感数据一旦收集并存储,存储敏感数据的数据库和服务器必须确保敏感数据的完整性和隐私性。

2. 数据完整性

一个安全的系统需要保证其存储的信息是有效的。云计算的数据完整性是指数据无论存储在数据中心,还是传输在网络中,均不会因为删除和修改而遭到破坏。对于云计算而言,由于超级权限的存在,完整性面临巨大的挑战。完整性指的是数据或者资源的可信度,通常使用防止非法的或者未经授权的数据改变来表达完整性。完整性包括数据完整性(即信息的内容)和来源完整性(即数据的来源,常称为认证)。信息的来源可能会涉及来源的准确性和可信性,也涉及人们对此信息所贼予的信任性。数据完整性通常包括以下方面:

(1) 系统和客体权限控制访问应用系统表和系统命令,因此只有授权用户能够改变数据;

(2) 参照完整性是指根据数据库中定义的各种规则,保证数据间一致性的关系;

(3) 数据库必须具备保护数据免受破坏性的病毒侵害的能力;

(4) 网络通信中必须保证数据不受删除、损坏和窃听。

3. 数据可用性

一个安全的系统必须确保授权用户无延迟的访问和使用相应数据。一般来讲,可用性是指对数据或者信息的期望使用能力。可用性是系统可靠性与系统设计中构一令重要方面,因为一个不可用的系统带来的后果可能还不如没有系统。可用性之所以与安全相关是因为攻击者会蓄意地使数据或者服务失效,以此来拒绝对数据或者服务的访问。企图破坏系统的可用性称为拒绝服务攻击,这可能是最难检测的攻击,因为这要求能够分析判断出某种异常的访问模式是否会带来对数据或者资源的蓄意破坏。

9.3.2　传统的数据安全威胁

传统的数据安全威胁如下：

1. 客户端的安全性

云计算的客户端主要有两种模式：浏览器模式，特定程序的模式。从本质上来讲两者都属于软件的范畴。目前，浏览器是主要的客户端。因此简单分析以下浏览器的安全性。目前市场上的浏览器有多种，它们主要有类：以微软 IE 为内核的，自己开发浏览器内核的。基于微软的 IE 内核浏览器是最普遍的浏览器。虽然浏览器技术不断进步，但是一个软件的 bug 是不可避免的。自从微软 IE 浏览器推出以来，发现了不少的安全漏洞。依据年份统计，微软 IE 浏览器自发布以来总共达到的安全漏洞数达 1810 个之多，在微软众多软件之中，是安全漏洞规模递增最快的软件之一。

2. 主从结构的安全性

云计算数据应用系统是典型的主从结构模式，其面临的主要威胁有：

1）拒绝服务攻击

DDoS 攻击往往采取合法的数据请求技术，再加上一些傀儡机器，造成 DDoS 攻击，这是目前最难防御的互联网攻击之一。传统的网络设备和周边安全技术，例如防火墙和 IDS、速率限制、接入限制等均无法提供非常有效的针对 DDoS 攻击的保护。它不断对网络服务系统进行干扰，改变其正常的作业流程，执行无关程序使系统响应减慢甚至瘫痪，影响正常用户的使用，甚至使合法用户被排斥而不能进入云网络或不能得到相应的服务。通过对主从结构特点进行分析，不难看出，拒绝服务攻击是对主从结构系统极其致命的一种攻击方式。一台应用服务器受到此类攻击不仅自身无法访问，还会使对数据中心的其他访问无法进行。拒绝服务攻击对一台主机或一个系统进行攻击时，表现初期仅仅是一台主机或一个系统访问缓慢或不能正常访问，随着攻击量的增加和持续，使访问连接数和流量达到数据中心当前最大阀值，从而使客户端不能正常维护和访问自己韵数据。

2）病毒传播

云计算通过互联网进行数据服务，因此无法阻止恶意者网络传播计算机病毒。它的破坏性大大高于单机系统，而且用户很难防范。现在互联网上绝大多数病毒更具有隐蔽性，且传播越来越快，像蠕虫王病毒不到一天就传遍全球；病毒制造技术持续提高。不仅结合黑客程序，可以强行破解和泄密，而且交叉感染，感染后形成病毒群集体攻击，因而病毒危害急剧增大。云计算的应用系统多为虚拟机上的 Linux 操作系统，Linux 是典型的开源操作系统，这就使内核级的，深隐藏的病毒木马更加防不胜防。

3）结点故障

云计算服务系统自 2008 年 7 月 8 日起频繁出错，目前几近崩溃。响应速度问题尤为突出，即使在大型局域网上响应速度也不容乐观。究其原因，主要是机器的故障引起的。数据中心结点的故障引起了用户数据的丢失和破坏不容忽视。因此结点的故障也是主从结构所面临的威胁之一。

3. 通信安全性

在网络环境中，通常将对安全的攻击分为两类：主动攻击和被动攻击。

被动攻击不会导致对网络传递的信息做任何改动，而且网络的状态也不被改变。因此被动攻击主要威胁信息的保密性，常见的被动攻击手段有：

（1）偷窃：用各种可能的手段窃取网络中的信息资源和敏感信息。

（2）分析：通过对网络自锯长期监视。应用统计分析等方法对诸如通信频度、通信流量以及信息流向等参数进行研究从丽得出有价值的信息。

主动攻击则意在篡改网络中传输的信息以达到攻击者的骨的。常见的主动攻击手段有：

（1）冒充：通过欺骗通信系统（或用户）达到非法用户冒充成为合法用户，或者特权小的用户冒充成为特权大的用户的目的。

（2）篡改：改变消息内容，删除其中的部分内容，用一条假消息代替原始消息，或者将某些额外消息插入其中。

9.3.3 新的数据安全威胁

对数据安全来讲，除了传统的安全威胁外，由于各种原因上述云计算的数据不可避免的带来新的安全威胁。新的数据安全威胁主要有以下几个方面：保密失效威胁，动态完整性威胁，分布式可用性威胁。

1. 保密失效威胁

传统的密码学的数据安全保护方式在云计算中不能直接采用，因为在云计算中，用户失去了对数据的控制。因此对于云计算中的数据完整性验证必须在没有完全数据知识的情况下进行。假设每个用户的许多不同的数据存储在云中并且必须确保用户的数据的进行安全验证，验证数据存储的准确性挑战性极大。

2. 动态完整性威胁

云计算不仅仅是数据仓库。云中的数据被用户不断的更新，包括插入，删除，修改等，因此，确保数据在动态环境中的安全是十分必要的。然而，这些动态特性使传统的完整性技术徒劳无功，必须有新的解决方案。

3. 分布式可用性威胁

云计算的发展是运行在同步性、一致性和分布式的数据中心的。用户数据的冗余存储在多个不同的物理空间中降低了用户的完整性威胁。因此，分布式的确保数据准确的存储协议对于构造一个健壮的，安全的数据存储系统十分的重要。

9.3.4 云计算的数据安全

数据安全意指通过一些技术或者非技术的方式来保证数据的访问是受到合理控制，并保证数据不被人为或者意外的损坏而泄露或更改。其包括以下三种数据安全类风险。

1. 访问控制风险

对于企业的数据访问权限，除企业用户外，云计算的运营商有数据优先访问权的，而企业的数据很可能面临被泄露的安全隐患。云计算服务商托管企业数据，但是数据的管理和安全由企业自己来承担负责。如果云计算服务商对于企业数据的仅仅存放，则表示企业客户无法对被托管数据加以有效利用。云计算厂商采用分权分级管理。为了防止云计算平台

供应商"偷窥"客户的数据和程序，可以采取分级控制和流程化管理的方法。银行是一个很好的例子，银行虽然存储着所有客户银行卡的密码，但即使是银行内部员工，也无法获取客户的密码信息；同时，银行系统内也有一系列流程防止出现"内鬼"。例如，将云计算的运维体系分为两级，一级是普通的运维人员，他们负责日常的运维工作，但是无法登录物理主机，也无法进入受控的机房，接触不到用户数据；二级是具备核心权限的人员，他们虽然可以进入机房也可以登录物理主机，但受到运维流程的严格控制。

2. 数据操作/管理风险

使用云计算服务，企业用户其实并不是很清楚自己的数据具体的托管服务器的位置以及具体是哪个服务器管理。增强安全防范意识。幸运的是，一点点常识和一些简单的正确电脑操作练习可以将这类安全性失误的影响降至最低，避免将你的机密资料放在云端上，如果你真的放了，例如利用网上银行时，避免在网络咖啡厅、学校或图书馆内的公用电脑上进行，也别太随便给出自己真正的联络资料，避免每个账号都使用同一个密码，就算只更改一个字母也好。云计算下增强安全意识，清楚地认识到风险，并采取必要的防范措施来确保安全。

3. 数据恢复风险

云计算中需要对数据进行有效的备份，保证在灾难发生的时候数据能及时地被恢复。

9.4 本章小结

9.1 节对对等网络的可信及公正性问题进行了研究，综述了当前研究中采用的激励机制、信任机制和安全机制来控制结点的自私行为和恶意行为。随着对等网络应用日益广泛，安全问题也将越来越突出，对对等网络的安全研究工作将更为系统和深入。

9.2 节对云计算平台的研究背景、研究动机阐述，综述了综述了当前研究进展现状。

9.3 节重点介绍了云计算的相关概念和虚拟化技术。从云计算本身的特点出发，分析了云计算的优势，介绍了几种云计算中的关键技术和安全问题。

参考文献

[1] Cornelli F，Damiani E，et al. (2002). Choosing reputable servents in a P2P network，ACM New York，NY，USA：376-386.

[2] Srivatsa M，Liu L. "Vulnerabilities and Security Threats in Structured Peer-to-Peer Systems：A Quantitative Analysis，" Proceedings of the 20th Annual Computer Security Applications Conference，2004.

[3] Adar E，Huberman B A. Free Riding on Gnetella. First Monday，5(10)，October 2000.

[4] Xiong L，Liu L. A reputation-base trust model for peer-to-peer eCommerce Communications. In ACM Conference(ACSAC 2004)，Tucson，Arizona，December 6-10，2004.

[5] John R Douceur. The Sybil attack. In Proceeding for the 1st International Workshop on Peer-to-Peer Systems(IPTPS'02)，Cambridge，Massachusetts，March 2002.

[6] Saroiu S，Gummadi P K，Gribble S D. A measurement study of peer-to-peer le sharing systems. In Proceedings of Multimedia Computing and Networking 2002 (MMCN)，San Jose，CA，USA，

January 2002.
[7] Kollock P. The Economies of Online Cooperation: Gifts and Public Goods in Cyberspace. Communicaties in Cyberspace, Chapter 9, 1999.
[8] Hardin G. The Tragedy of the Commons. Science. 1968 Dec 13; 162(5364):1243-8
[9] Feldman M, Lai K, Stoica I, Chuang J. Robust Incentive Techniques for Peer-to-Peer Networks. ACM E-Commerce Conference (EC'04), May 2004.
[10] http://www.kazaa.com.
[11] MojoNation, http://www.mojonation.net.
[12] Kung H T, Wu C H. Differentiated Admission for Peer-to-Peer Systems: Incentivizing Peers to Contribute Their Resources. 2003.
[13] Golle P, Leyton-Brown K, Mironov I. Incentives for Sharing in Peer-to-Peer Networks. in ACM Conference on Electronic Commerce. 2002.
[14] Yang B, Garcia-Molina H. PPay: Micropayments for Peer-to-Peer Systems. In Proceedings of the 10th ACM Conference on Computer and Communications Security(CCS), oct. 2003.
[15] Ioannidis J, Ioannidis S, et al. Fileteller: Paying and Getting Paid for File Storage, Springer-Verlag; 1999:282-300.
[16] Vishnumurthy V, Chandrakumar S, et al. Karma: A secure economic framework for p2p resource sharing. 2003.
[17] Hausheer D, Stiller B. PeerMint: Decentralized and Secure Accounting for Peer-to-Peer Applications, Springer. 2005.
[18] Dabek F, Kaashoek M F, et al. Wide-area cooperative storage with CFS, ACM New York, NY, USA. 2001, 35:202-215.
[19] Adya A, Bolosky W J, et al. FARSITE: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment, ACM ASSOCIATION FOR COMPUTING MACHINERY. 2002, 36:1-14.
[20] Cox L P, Murray C D, et al. Pastiche: making backup cheap and easy, ACM New York, NY, USA. 2002, 36:285-298.
[21] Cox L P, Noble B D. Samsara: honor among thieves in peer-to-peer storage, ACM New York, NY, USA. 2003:120-132.
[22] Sun Q, Garcia-Molina H. SLIC: a selfish link-based incentive mechanism for unstructured peer-to-peer networks. 2004:506-515.
[23] Axelrod R, Hamilton W D. The evolution of cooperation. 1981, 211:1390-1396.
[24] Cohen B. Incentives Build Robustness in BitTorrent, Berkeley, CA, USA. 2003, 6.
[25] Riolo R L, Cohen M D, et al. Evolution of cooperation without reciprocity. 2001, 414:441-443.
[26] Hales D, Edmonds B. Applying a socially-inspired technique (tags) to improve cooperation in P2P Networks. IEEE Transactions in Systems, Man and Cybernetics - Part A: Systems and Humans, 2005, 35(3):385-395.
[27] Kamvar S D, Schlosser M T, et al. (2003). The Eigentrust algorithm for reputation management in P2P networks, ACM Press New York, NY, USA. 2003:640-651.
[28] Luhmann N. Trust and Power. Chichester: Wiley. 1979.
[29] Gambetta D. Can we trust trust? In: Gambetta D, ed. Trust: Making and Breaking Cooperative Relations. Basil Blackwell: Oxford Press, 1990. 213-237.
[30] Randison T, Sloman M A Survey of Trust in Internet Applications. 2000, 3:2-16.
[31] Chen R, Yeager W Poblano: A distributed trust model for P2P networks. Technical Report, TR-I4-02-08, Palo Alto: Sun Microsystem, 2002.
[32] Alfarez Abdul-Rahman, Stephen Hailes, Supporting Trust in Virtual Communities, Proceedings of the

33rd Hawaii International Conference on System Sciences-Volume 6, p. 6007, January 04-07, 2000.
[33] Garfinkel S. PGP: Pretty Good Privacy, O'Reilly. 1995.
[34] Wang Yao, Julita Vasileva. Trust and Reputation Model in Peer-to-Peer Networks. P2P'03, 2003.
[35] Wang Yao, Vassileva Julila. Bayesian network model in peer-to-peer networks Agents and Peer-to-Peer. Computing AP2PC 2003, 2003, p23-34.
[36] Josang A. Trust-Based Decision Making for Electronic Transactions, Proceedings of the Fourth NoSDic workshop on Secure Computer Systems (NOSDSEC'99). Stockholm University, Sweden, 1999.
[37] Abdul-Rahmma A, Hailes S. Supporting Trust in Virtual Communities In Proceedings Hawaii IntemationaI Conference on System Sciences 33, Maui, Hawaii, 4-7 January 2000.
[38] Damiani E, Vimereati D C, Paraboschi S, et al. A Reputation-based approach for Choosing Reliable Resources in Peer-to-Peer Networks. In Proc. of the 9th ACM Conference On Computer and Communications Security. 2002.
[39] Kamvar S D, Schlossc M T. EigenRep: Reputation Management in P2P Networks. The 12th International World Wide Web ConferenCe. Budapest, Hungary: ACM press. 2003.
[40] 窦文,王怀民,贾焰等. 构造基于推荐的 Peer-to-Peer 环境下的 Trust 模型. 软件学报. 2004,15(4):571-583.
[41] 宋雪吕,陆建德. 基于组群的 P2P 系统中的信誉机制[J]. 微机发展. 2005,15(11):11～13.
[42] Gummadi A, Yoon J P. Modeling group trust for peer-to-peer access control. In Proceedings of the 15th International Workshop on Database and Expert Systems Applications (DEXA'04). 2004.
[43] Christin N, Weigend A S, et al. Content availability, pollution and poisoning in file sharing peer-to-peer networks, ACM New York, NY, USA. 2005:68-77.
[44] Liang J, Kumar R, et al. Pollution in P2P file sharing systems. 2005.
[45] Daswani N, Garcia-Molina H, et al. (2003). Open Problems in Data-Sharing Peer-to-Peer Systems, Springer:1-15.
[46] 左敏,李建华. P2P 中的文件污染与污染防治,计算机工程,2007,38(18).
[47] Overpeer spreads fake files through P2P networks, http://www. afterdawn. com/news/archive/3101. cfm.
[48] Kenji Leibnitz, Tobias Ho? feld, Naoki Wakamiya, and Masayuki Murata. On Pollution in eDonkey-like Peer-to-Peer File-Sharing Networks, 13th GI/ITG Conference on Measurement, Modeling, and Evaluation of Computer and Communication Systems (MMB 2006), Nürnberg, Germany, March 2006.
[49] Pouwelse J, Garbacki P, et al. The Bittorrent P2P File-Sharing System: Measurements and Analysis, Springer. 2005, 3640:205.
[50] Gu Q, Bai K, et al. Modeling of pollution in p2p file sharing systems. 2006.
[51] Thommes R, Coates M Epidemiological Models of Peer-to-Peer Viruses and Pollution. 2006.
[52] Lee U, Choi M, et al. Understanding Pollution Dynamics in P2P File Sharing. IPTPS06.
[53] Walsh K, Sirer E G. Thwarting P2P Pollution Using Object Reputation. 2005.
[54] Gannon D. Head in the cloud[J]. nature. 2007.
[55] 刘鹏. 云计算发展现状[EB/OL]. http://www. chinacloud. cn/show. aspx? id=754&cid=11.
[56] Morgan S. Technology Trends. 2008.
[57] Gartner. http://www. gartner. eom/DisplayDocurnent? id-685308, 2009.
[58] 李开复. 云计算[EB/OL]. 十七届国际万维网大会报告. 2009. 6.
[59] 欧阳璟. 云计算,未来的趋势[EB/OL]. 程序员. 2008. 11:38-41.
[60] Amazon Elastic Compute Cloud. http://aws. anlazon. eom/ec2.

[61] Hamilton D. Cloud computing seen as next wave for technology investors[EB/OL]. Financial Post. 04 June 2008. http://www. financialpost. eom. money. story. html? id=562877.
[62] Francesco M, Gianni F. An approach to a cloud Computing network[J], IEEE, August 2008:113-118.
[63] Micheal A, Armando F, Rean G. Above the clouds: A Berkeley View of Cloud Computing. Technical Report, UCB/EECS-2009—28. University of California at Berkeley. February 10, 2009.
[64] Weiss A. Computing in the Clouds[J]. networker. ACM. 2007, 11 (4): 15-25.
[65] Stephen H. Cloud Computing's Security[EB/OL]. 通讯世界. 2009. 11.
[66] Siani Pearson, Yun Chen. A Privacy Manager for Cloud Computing[EB/OL]. HPlabs.
[67] http://www. RSAconference. corn/2010/usa/agenda-and-sessions. htm.
[68] http://netsecurity. 51cto. tom/art/201003/186066. htm.
[69] Heiser J, Nicolett M. Assessing the Security risks of coud computing. http://www. gartner. com/DisplayDocument? id=685308, 2008.
[70] A Chronology of Data Breaches Since the ChoicePoint Incident. http://www. privacyrights. org/ar/ChronDataBreaches. htm.
[71] Windows vista voice recognition command execution vulnerability. http://www. SeoJrityfoClls. eom/bid/22359, 2007.
[72] Linux kernel netfiler do_replace local buffer overflow vulnerability. http://cve. mitre. org/cgi-bm/cvename. cgi? name=CVE-2006-0038.
[73] Efstathopoulos P, Krolm M. Make least privilege a right (not a privilege) [C]. Proceedings of the Proceedings of USenix Workshop on Hot Topics in Operating Systems. 2005.
[74] Loscocco P. Intergrating Flexibility Support for Security Policies into the Linux Operation System [c]. 2001 USENIX Annual Technical Conference table of contents. USENIX Association Berkeley CA USA, 2001: 29-40.
[75] Li N, Mao Z, Chen H. Usable Mandatory Integrity Protection for Operation Systems[C]. Proceedings of the IEEE Symposium on Security and Privacy. 2007: 165. 177.
[76] Petreni Jr N, Hick M. Automated detection of persistent kernel control flow attacks[C]. Proceeding of the 14th ACM conference on computer and communications security. ACM New York. 2007: 104-113.

CHAPTER 10

第 10 章 总结与展望

10.1 对等网络的总结与展望

10.1.1 对等计算总结

如何定位资源和查找资源是 P2P 系统应用所要解决的核心问题，它是由拓扑、路由和查询这三个紧密相关的设计技术所决定的。本书紧密围绕此核心问题，结合最新研究前沿，针对 DHT 机制的本质，从拓扑、路由和查询这三个方面展开研究，主要的研究内容如下：

(1) 提出了在动态网络环境下自适应的拓扑调整模型(Session Heterogeneity Topology，SHT)，能够较好地控制 DHT 拓扑维护开销并提高了系统资源的可用性，有效解决了 DHT 拓扑维护开销问题。SHT 利用结点的会话异构特性将 DHT 重构，将 DHT 的拓扑构造变成仅由稳定结点组成，从而大大降低了拓扑适应动态环境的调整开销。研究揭示出会话时间短的结点是拓扑调整的主要扰动因子，因此建议拓扑设计时将它们聚簇于稳定结点，并由稳定结点代理它们的请求，从而在保证它们能够正常使用系统的同时又将它们的扰动限制于局部稳定结点的动荡。理论分析和实验表明，一个合理的聚簇大小存在，使得拓扑达到近似最优化，此时减少的开销达到近似最小。

(2) 提出了基于小世界理论的概率缓存链技术，通过附加少量开销，能够较大地减少 DHT 路由跳数。理论分析和仿真实验表明在保持高度分散的路由低状态下，可实现较高的路由效率，为路由的状态与效率折中问题提供了一个可行的高效的解决方案。据此设计的 PCCAN 系统，在保持 $O(1)$的路由状态下将 CAN 的路由路径长度由 $O(n^{1/d})$改进到 $O(\log^2(n^{1/d}))$。研究利用了 DHT 路由的贪婪特性及查询的反馈机制，构造了概率缓存的路由快捷链，由此在路由表链间形成了一个小世界，使得系统的路由性能得到全局性的优化。设计充分考虑了静态与动态下的收敛问题，并设计主动查询机制以适应动态环境并加速收敛，使系统性能得到保障。由于设计采用的是查询反馈机

制以及缓存技术，具有较强的现实意义。特别对于当前的一大类采用低状态 DHT 路由表的具有高度松散结构的 P2P 系统，能够在轻微的修改下达到路由效率的可观的提高。

(3) 提出了基于向量空间模型(VSM)的相似文档搜索方法和技术，使得 DHT 查询能支持多关键字查询和相似文档搜索，从而突破了 DHT 查询的单关键字的精确匹配约束，使 DHT 查询的应用范围大为扩展，有效解决了 DHT 查询的应用局限性问题。研究在两方面对 DHT 查询进行了改进。一方面，目前基于 DHT 的多关键字查询技术，通常采用单关键字的组合查询的模式，带来较多的网络开销。通过 VSM 技术将多个关键字形成了向量空间模型 VSM 中的向量，并使得查询相关于此向量，从而能够支持多关键字查询并去除了单关键字组合查询所需的大量网络开销。另一方面，当前 DHT 查询的局限性本质是由于哈希使得查询失去了语义性支持，仅能够支持精确匹配。而利用 VSM 技术进行文档标识的哈希空间重映射，使得 DHT 文档标识具有相似性的语义，从而支持相似性文档搜索。

(4) 在框架模型中的 overlay 层，针对 Chord 协议进行深入研究：首先，在 Chord 满环情况下，对 Chord 路由进行了形式化描述，将路由过程抽象成一个整数由一个数列受限的线性表示问题，给出了 Chord 协议的最优路由表结构，并证明了在满环情况下三倍数 Chord 路由表是最优路由表；其次，当 Chord 非满环情况下，提出了对 Finger 表进行重新构造的方法，实验表明，该重构有效地提高了 Overlay 层的路由效率。

10.1.2　P2P 技术展望

P2P 系统中 DHT 机制应用是个广泛而重要的研究课题，本书的工作只是初步地解决其中的一些问题，在本书基础上可进一步开展的工作包括：

(1) 路由的响应时延。本书的小世界模型引入在 Overlay 层上达到了性能的优化，进一步的问题是：如何在 Overlay 层优化的同时又保证路由的位置，能否建立相关理论模型。小世界路由提供了更广泛的邻居路由结点，有望为这个问题提供良好的切入点。

(2) 拓扑的融合。本书研究的控制拓扑开销的思想在于渐近构造一个由稳定结点组成的 DHT 拓扑，其本质是一种混杂拓扑。进一步可以考虑将非结构化拓扑与结构化拓扑融合。因为非结构化拓扑能够较好地适用于动态环境，而结构化拓扑能够保证良好性能，因而，这两种主流拓扑的融合是研究的一个有趣的方向。发表在最近的 IPTPS04 会议的文章[1]提出的观点是：在非结构化为底层的 Gnutella 网络中形成一个 DHT 小社区。这在一定程度启示了这种融合的可行性，但仍然有大量的挑战性工作有待研究。

(3) 查询的 DHT 标识语义。本书的研究将信息检索领域中的 VSM 和 DHT 技术相结合，把相似性语义融入 DHT 标识中，从而支持多关键字查询和相似文档搜索。注意到 DHT 查询是以 DHT 标识为查询关键字的，当前这种哈希化的 DHT 标识仅能够支持单关键字的精确匹配，但我们将相似语义加入后则可以支持多关键字的查询。事实上，DHT 标识的语义对于 DHT 查询来说是非常重要的，是对传统的 DHT 查询方式的一种重大变革，因而是一个非常有意义的研究方向。本书提供了对 DHT 标识语义的初步探索，进一步的研究可以考虑利用各种哈希语义模型以及结合信息检索的新技术来探索 DHT 标识的更丰富的语义支持。如考虑让 DHT 标识提供层次化语义从而直接支持层次化的目录文件 locality 和让 DHT 标识提供位运算语义从而直接支持多关键字的组合查询。

今后在 P2P 信息检索领域的进一步研究工作可以从以下这几个主要方面入手：

(1) 基于机器学习来逐步理解用户偏好的 P2P 信息检索系统。可以从用户查询的偏好、上下文环境发现用户的查询目标文档范围和分类,并在该分类上做进一步的查询、检索和多次反馈,并能快速地、高质量地返回用户希望的结果。该系统也能够直接根据查询词精确判断所属的类别范畴,并在生成的局部专题中查找,提供给查询用户。这样同样可以提高主题挖掘的深度和准确性。

(2) 基于统计加语义的对等网络信息检索。可以利用自然语言来进行查询、检索,这个方面需要解决如文本挖掘(Text Mining)和自然语言处理(NLP)等关键技术。另外,在 P2P 文档相似性判断过程中所使用的向量空间模型表示方面,可以同时结合计算语言学、自然语言处理等相关知识,使用概念空间代替目前使用的词空间,用来提高检索精度,改善信息检索质量。随着 NLP 技术的发展和成熟,利用人类语言查询将成为重要的信息获取手段。对数据的推理和联想成为可能,基于语义的信息检索和知识发现将成为下一代对等网络信息检索的重要研究方向,也是重大的挑战。

(3) 在 P2P 网络中,从多个结点分布式地检索出各自的结果然后进行结果排序和合并,也是急需解决的一个问题。

10.2 网格计算的总结与展望

10.2.1 网格计算总结

随着计算机技术和网络技术的发展,以及日益增长的计算力需求,诞生了网格计算。构建一个网格系统,需要研究信息服务、数据管理、安全机制等技术。同时,在网格计算中,资源管理对高效合理利用计算资源起着十分重要的作用。网格资源具备动态性、异构性和自治性的特征,需要对网格资源管理和调度的关键技术作相关研究。网格资源的动态性、自治性和异构性,使得网格资源提供者和使用者具有对等网络的特点,因此有必要将对等网络方法引入到网格资源管理中。

本书的网格计算部分首先介绍了网格资源管理和调度策略的一些基本概念和主要的研究内容,然后结合网格自身特点和发展趋势,对其中的几个关键问题进行了深入的研究,包括网格系统的体系结构,网格资源信息的表示方法、资源管理和任务调度算法以及负载均衡。为了验证本书提出的模型和算法的有效性,基于一个网格实验平台 DDGrid——新药研发网格,我们进行了大量的实验,实验结果证明了本书提出的模型和算法的有效性。

本书的网格计算部分主要研究了以下几个方面:

(1) 在分析了网格自身特点和发展趋势的基础上,将对等网络方法引入网格的资源管理和调度,设计了基于超级结点对等网络的网格资源管理体系结构。这种集中式和分布式的混合结构设计,能够解决现有网格系统采用的集中式管理容易引起的单点失效、性能瓶颈等问题,从而可以更好地描述网格资源的动态性、自治性等特点,使网格系统具有更强的鲁棒性和自适应性,并且有利于制定优化网格资源管理和调度的策略、算法。进一步,根据网格资源提供者的 IP 层信息生成含有路由信息的 Overlay Network 拓扑,并且使用有向图表示该拓扑结构。这种使用有向图进行网格拓扑结构表示的方式在能够准确描述网格资源提供者的计算能力的同时,还能够弥补其他现有的资源信息表示模型的 Overlay 层路由信息

不能精确反映IP层路由情况的不足。同时这种简单的描述方式有利于网格资源调度器发掘网格资源提供者和网格任务之间的对应关系。

(2) 提出了基于树匹配的nTreeMatch算法。算法结合DAG图的任务表示形式，通过树形数据结构匹配的方法解决了网格资源和网格任务间的映射问题。同时算法充分利用overlay拓扑中结点的路由信息，以轻量附加开销来有效减少overlay层上的路由跳数，使得overlay层上的路由跳数尽量接近IP层上的路由跳数，降低RDP。理论和模拟实验表明在大规模的网格系统中，算法在进行资源调度时可以获得较高的路由效率，为路由的状态与效率折中问题提供了一个可行的解决方案。该算法尤其适用于为特定的科学应用而设计的专用计算网格的资源调度。

(3) 针对基于Web Service的通用网格系统的资源调度，提出了基于资源发现的GChord算法。考虑到网格的动态性特征，GChord算法采用服务发现的方式解决资源调度问题，将资源需求按照Chord路由协议在网格中转发，改变了传统的集中式调度方法采取的信息收集方式，能够实时反映网格结点的工作负载状态，有效解决由于信息过时、数据不一致而引起的任务再调度问题。实验证明，GChord算法可以实现网格系统的实时资源调度，并且使得网格系统保持良好的负载均衡状态。

(4) 为解决网格资源调度中动态负载均衡的挑战，在研究了多代理技术和网格计算相互融合的发展趋势的基础上，提出了基于多代理协同计算的rwAgent算法。算法利用多代理技术，通过代理的自治性和智能学习，实现网格资源的分散调度，同时可以获得很好的负载均衡效果。严格的数学建模和理论分析证明，rwAgent算法可以实现资源调度过程中网格系统的全局负载均衡，实验结果证明了算法的有效性和优越性。

10.2.2 网格计算展望

随着网格技术的发展，用户对网格系统的服务质量提出了更高的要求，因此，网格资源管理和调度的研究领域出现了新的挑战。本书在网格资源管理方式和并行任务调度上做了一定的研究，但要完全满足人们对网格系统的需求，还有许多地方值得进一步研究，具体如下：

(1) 拓扑对于网格系统性能有重要的影响，直接相关于网格资源管理和调度算法的路由性能，因此网格系统地拓扑结构应该能够适应动态的网格环境的变化，并与物理网络紧密结合。本书使用类似超级结点的P2P网络来表示网格系统的Overlay Network拓扑结构，而现在P2P的拓扑研究又有了新的进展，例如：控制拓扑维护的开销，来增强拓扑对于网络的动态适应性；克服DHT结构的平坦化、加入现实中的层次特征的层次式拓扑结构；结合结构化和非结构化两种拓扑优点的混合式拓扑结构。因此，未来的工作中我们将对于新型P2P拓扑结构如何和网格系统相结构进行研究，进一步提高网格资源管理的有效性。

(2) 本书的网格计算部分在网格资源管理和调度算法方面的研究工作均基于一个前提，即物理网络的性能是预先确定的。例如使用系统的benchmark信息将网格结点可提供的计算资源量化为一个确定的值，在未来的研究工作中，我们将考虑开发网络性能预测工具包，更精确、实时地表示物理网络的性能。

(3) 本书的网格计算部分的rwAgent算法使用多代理协同计算计算技术进行网格资源的动态调度。在未来的工作中，我们会考虑建立代理知识库或者使用蚂蚁克隆技术，使得代

理可以根据以前的先验知识，快速找到满足要求的网格结点，以缩短任务需求和可利用资源间的匹配时间，以此来提高 rwAgent 算法的效率。

(4) 参与网格计算的网格资源除了本书考虑的存储空间、CPU、内存等因素之外，还包括网络带宽、链路延迟等因素，因此，在未来的工作中，我们还将考虑这些因素对资源调度的影响。

10.3　云计算的总结与展望

云计算是个很热的话题，现在可以说我们将要进入了云计算的时代。人们高度重视云计算的原因，一方面是云计算这种商业模式孕育着巨大的商业机遇，另一方面，云计算作为互联网资源的新的配置方式，它将改变整个互联网的价值链。

10.3.1　云计算总结

分布式文件系统(DFS)是一种网络文件系统，它分布在多个计算机结点上，每个结点只会直接存取整个文件系统的一部分。我们参考了两个比较成熟的 DFS 架构——GFS 和 Hadoop，设计并实现了 CFS 文件系统。CFS 是一个面向数据密集型应用的分布式文件系统，整个系统由 Master 结点、数据结点和客户端组成，具有一定的容错性和可扩展性。测试和实验表明，CFS 具备了文件系统的基本功能，实现了对数据的流式读写操作，并有一定的容错功能。

借鉴了 GFS 和 HDFS 的设计思想，根据测试和实验，系统实现了分布式文件系统的一些基本功能，具备一定的容错性，并达到了理想的运行效率。按照最初的设计，CFS 应能屏蔽操作系统和硬件的异构性，实现跨平台的系统。

10.3.2　云计算展望

随着云计算模式的进一步发展成熟，还有许多云计算的问题期待进一步研究，具体如下：

(1) 云平台数据安全性问题。保存在"云"中的文件可能包含私人数据。当前的云从本质上来说是提供了公共(而不是私有)网络，因此会遭受更多的攻击，用户需要更多的保证，而不仅仅是让这些公司简单宣称"我们已经隐藏了您的数据"。这也就是为什么云计算的安全性至今仍被多数的企业和用户所质疑。为此，在 Gartner 发布的一份名为《云计算安全风险评估》的报告中，列出了云计算技术存在的七大风险，即特权用户的接入、可审查性、数据位置、数据隔离、数据恢复、调查支持和长期生存性。

(2) 服务的可用性和性能的不可预知性。现今，为了提高可用性，解决方案就是多云计算，还有，可用性相关的另一个问题是 DDoS(Distributed Denial of Service，分布式拒绝服务)攻击。由于弹性，云计算将攻击目标从 SaaS 提供商转移到能够马上吸引攻击并且具有 DDoS 攻击保护能力的效用计算提供商。还有一个待解决的问题是性能的不可预知性。

(3) 声誉和法律危机。一个用户的恶意操作会影响到整个"云"的声誉。创建类似于信任邮件服务那样的声誉保护服务将可能会成为一个机遇。另外一个问题是法律责任的转移——当出现问题时，云计算提供商将会希望由用户去承担相应法律责任，而不要将责任转

嫁给他们。当前软件许可证通常限定在运行软件的机器上,用户购买软件并按年支付维护费用。许多云计算提供商从一开始就倾向于开源软件,这正是因为商业软件许可证模式并不适合效用计算。首要的机遇要么开源持续流行,要么商业软件公司改变他们的许可证结构,让其更加适合云计算。还有一个办法是鼓励软件公司制定销售政策来向云计算提供产品。将软件公司销售部门的一些反对派转到支持云计算者的阵营中来。

参考文献

Loo B T, Ryan Huebsch, Ion Stoica, Joseph M Hellerstein. The Case for a Hybrid P2P Search Infrastructure. Proceedings of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS'04), Feb 2004.